5 luglio 2017 - laboratorio

Constraint Programming

Prof. Marco Gavanelli

5 luglio 2017

Descrizione problema

In molti sistemi operativi (ad es. le distribuzioni Debian di Linux) i software installabili sono suddivisi in pacchetti. Ogni pacchetto disponibile può essere installato o non installato nel sistema.

Le informazioni sui pacchetti sono riportate in un file `pack_inst.ecl` (per ECLiPSe) e in un file `pack_inst.asp` (per ASP); ciascun pacchetto è rappresentato con un identificatore numerico.

Esistono pacchetti che sono in conflitto con altri pacchetti; due pacchetti in conflitto non possono essere installati entrambi. Nei file `pack_inst.ecl` e `pack_inst.asp` sono riportati dei fatti `conflict(X,Y)`, che indicano che i pacchetti *X* e *Y* sono in conflitto. Ad esempio, se è presente il fatto `conflict(2,4)` significa che il pacchetto con identificatore 2 ed il pacchetto con identificatore 4 non possono essere entrambi installati: se ne può installare uno dei due o nessuno, ma non entrambi.

Alcuni pacchetti dipendono da altri pacchetti e quindi ne richiedono l'installazione. Un fatto `requires(X,Y)` indica che se viene installato il pacchetto X, allora si deve installare anche il pacchetto Y (mentre non è detto che installando Y si debba installare anche X).

Nel momento in cui l'utente richiede l'installazione di un pacchetto, è quindi possibile che debbano essere installati alcuni pacchetti e disinstallati altri. L'utente ha richiesto l'installazione dei pacchetti indicati dai fatti `install(X)`.

Si desidera trovare il numero minimo di installazioni e disinstallazioni di pacchetti in modo che tutti i vincoli (conflitti e dipendenze) siano rispettati.

CLP (15 punti)

Si risolva il problema usando ECL*i*PS*e*. Nel file `pack_inst.ecl` sono riportati:

- un fatto `package(Pacchetti)`, dove `Pacchetti` è una lista che contiene tutti gli identificativi di pacchetti che possono essere installati
- un fatto `installed(Installati)`, dove `Installati` è una lista della stessa lunghezza della lista `Pacchetti`; l'N-esimo elemento della lista `Installati` è un valore 0 o 1;
  - 1 indica che il pacchetto nell'N-esimo elemento della lista `Pacchetti` è iniziamente installato
  - 0 indica che non è inizialmente installato
-

ASP oppure MiniZinc (10 punti)

Si risolva il problema in Aswer Set Programming oppure in MiniZinc.

- nel file `pack_inst.asp` vengono forniti:
  - un predicato `package(P)` che è vero per tutti i valori `P` che sono pacchetti
  - un predicato `installed(I)` che è vero per tutti i valori di `I` che sono pacchetti inizialmente installati nel sistema
-
- nel file `pack_inst.mzn` (da copiare e incollare nel sorgente) vengono forniti:
  - `int n`: numero di pacchetti
  - `requires`: una matrice `Nreq × 2`; ogni riga della matrice è una coppia di pacchetti; la coppia [X,Y] significa che il pacchetto X richiede l'installazione del pacchetto Y
  - `conflict`: una matrice *Nconf* ×2; ogni riga della matrice è una coppia di pacchetti; la coppia [X,Y] significa che i pacchetti X e Y sono in conflitto e quindi non possono essere entrambi installati nel sistema
  - `install`: un array di `n` elementi; `install[i]` vale 1 se il pacchetto `i` deve essere installato; 0 se non è necessario installarlo
  - `installed`: un array di `n` elementi; `installed[i]` vale 1 se il pacchetto `i` è stato installato; 0 se non è inizialmente installato

Soluzioni

Soluzione CLP

```
:- lib(fd).
:- lib(fd_global).
:- lib(listut).
:- [pack_inst].

pip(Ldec):-
 findall([P1,P2],requires(P1,P2),Lreq),
 installed(Linst),
 package(Lpack),
 findall([P1,P2],conflict(P1,P2),Lconf),
 findall(P,install(P),Li),
 length(Lpack,Npack),
 length(Ldec,Npack),
 Ldec :: 0..1,
 install_constraint(Li,Ldec),
 require_constraint(Lreq,Ldec),
 conflict_constraint(Lconf,Ldec),
 objective(Linst,Ldec,Lcosts),
 fd_global:sumlist(Lcosts,F),
 minimize(labeling(Ldec),F).

install_constraint([],_).
install_constraint([H|T],LD):-
 nth1(H,LD,1),
 install_constraint(T,LD).

require_constraint([],_).
require_constraint([[A,B]|T],Ldec):-
 nth1(A,Ldec,VA),
 nth1(B,Ldec,VB),
 VA #=< VB,
 require_constraint(T,Ldec).

conflict_constraint([],_).
conflict_constraint([[A,B]|T],Ldec):-
 nth1(A,Ldec,VA),
 nth1(B,Ldec,VB),
 VA + VB #=< 1,
 conflict_constraint(T,Ldec).
```

```
objective([],[],[]).
objective([A|LA],[B|LB],[C|LC]):-
 A #\= B #<=> C,
 objective(LA,LB,LC).
```

Soluzione MiniZinc

```
array[1..n] of var 0..1: x;

constraint forall(i in 1..Nconf)
 (    x[conflict[i,1]] + x[conflict[i,2]] < 2  );

constraint forall(i in 1..n)
 ( x[i] >= install[i]);

constraint forall(i in 1..Nreq)
 ( x[requires[i,1]] <= x[requires[i,2]]);

solve minimize sum([abs(x[i]-installed[i])|i in 1..n]);
```

Soluzione ASP

```
{install(P)} :- package(P).

:- install(P1), requires(P1,P2), not install(P2).

:- conflict(P1,P2), install(P1), install(P2).

change(P):- package(P), install(P), not installed(P).

change(P):- package(P), not install(P), installed(P).

#minimize { 1,P:change(P)}.

#show install/1.
#show change/1.
```