

Programmazione Dichiarativa

Jacopo Castellini

A.A. 2015/16

Testo del progetto

Si consideri un tragitto percorso da un autobus. Si può considerare il tempo trascorso durante il tragitto come discretizzato. Ovvero come se il tragitto fosse composto da una successione di punti significativi (per es. le fermate).

Identifichiamo i punti significativi con i numeri $P = \{1, 2, 3, \dots, K\}$. Vi sono punti "significativi" di due tipi:

*) le fermate (un insieme di punti F sottoinsieme di P)

*) i checkpoint (l'insieme dei punti che non sono fermate: $C = P - F$)

Alle fermate l'autobus si ferma e possono salire e/o scendere delle persone. I checkpoint non sono fermate e nessuno può salire o scendere, ma tra due fermate consecutive ci possono essere uno o più checkpoint.

Supponiamo di rappresentare anche le persone come numeri interi e che per ogni istante j in C sia X_j l'insieme (incognito) delle persone a bordo dell'autobus. Per semplicità supponiamo anche che se una persona scende, poi non risale più.

Supponiamo anche che per ogni checkpoint k in C si sappia che l'insieme delle persone X_k è incluso in un insieme Y_k . (Ovvero Y_k è una stima per eccesso di quali siano le persone a bordo: Y_k include sicuramente tutti quelli che sono a bordo ma potrebbe contenerne anche altri "estranei").

Si noti che se j , ed h , sono due checkpoint tra due fermate consecutive (cioè tra j e h non c'è alcuna fermata) allora necessariamente $X_j = X_h$, inoltre Y_j include X_j e Y_h include X_h . Ma Y_j e Y_h possono essere diversi.

Problema consiste nel determinare tutti gli insiemi X_j incogniti (per ogni j in C) conoscendo solo gli insiemi Y_r (per ogni r in C). Dato che ci potrebbe essere più di una sola soluzione (per esempio la soluzione banale in cui tutti gli X_j sono vuoti e' ammissibile), si vuole identificare la soluzione che minimizza le cardinalità degli insiemi $Y_j - X_j$. Ovvero quella "ottimista" che considera che le stime siano più affidabili possibile.

Implementazioni

Abbiamo fornito la soluzione al problema sia nel linguaggio ASP sia nel linguaggio CLP.

La soluzione in ASP (file *solver_asp.lp*) è la seguente:

```
{xj(J,P):checkpoint(J),person(P)}:-yj(J,P).  
  
:-xj(J,P),checkpoint(J),person(P),J==I+1,checkpoint(I),not xj(I,P).  
:-xj(J,P),checkpoint(J),person(P),J==I-1,checkpoint(I),not xj(I,P).  
  
#maximize{xj(J,P):checkpoint(J),person(P)}.  
#hide.  
#show xj/2.
```

L'approccio utilizzato è di tipo generate and test, cioè vengono prima generate tutte le possibili soluzioni e poi ne viene testata l'effettiva validità. Tramite un ground cardinality constraint vengono generati tutti i possibili atomi X_j per cui esiste il corrispondente Y_j (fase generate), mentre i due

vincoli sotto servono ad escludere gli atomi che non sono effettivamente corretti (fase test), cioè quelli che indicano una persona P presente in un checkpoint J ma non in quelli adiacenti J+1 e/o J-1 (poiché se due checkpoint i e j sono vicini allora deve valere $X_i = X_j$). L'istruzione maximize calcola l'answer set che massimizza la cardinalità dell'insieme contenente gli atomi X_j (poiché massimizzare quello è come minimizzare gli insiemi $Y_j - X_j$), mentre le ultime due righe servono a presentare la soluzione.

La soluzione CLP (file *solver_clp.pl*) invece è la seguente:

```
:-use_module(library(clpfd)).

checkpoint(X,L):-element(_,L,X).

not_present(X,Y,L):-get_people(X,L,R),all_different([Y|R]).

get_people(_,[],[]).
get_people(X,[[Y,Z]|T],R):-X==Y,get_people(X,T,R1),R=[Z|R1].
get_people(X,[[Y,_]|T],R):-X\=Y,get_people(X,T,R).

solve(C,L,R):-length(L,N),length(R,N),R ins 0..1,constraints(C,R,R,L,L),sum(R,#=,V),labeling([max(V)],R).

constraints(_,[],_,[]).
constraints(C,[A|R],Z,[[X,Y]|T],L):-check(C,A,Z,X,Y,L),constraints(C,R,Z,T,L).

check(C,A,Z,X,Y,L):-J is X+1,checkpoint(J,C),not_present(J,Y,L),V is X+1,constraint_prev(V,Y,Z,L,Z,L),W is X-1,constraint_next(W,Y,Z,L,Z,L),A#=0.
check(C,A,Z,X,Y,L):-J is X-1,checkpoint(J,C),not_present(J,Y,L),V is X+1,constraint_prev(V,Y,Z,L,Z,L),W is X-1,constraint_next(W,Y,Z,L,Z,L),A#=0.
check(_,_,_,_,_).

constraint_next(_,_,[],_,_).
constraint_next(X,Y,[A|_],[[Z,W]|_],P,T):-X==Z,Y==W,A#=0,V=X+1,constraint_next(V,Y,P,T,P,T).
constraint_next(X,Y,[_|R],[[_,_|L],P,T):-constraint_next(X,Y,R,L,P,T).

constraint_prev(_,_,[],_,_).
constraint_prev(X,Y,[A|_],[[Z,W]|_],P,T):-X==Z,Y==W,A#=0,V=X-1,constraint_next(V,Y,P,T,P,T).
constraint_prev(X,Y,[_|R],[[_,_|L],P,T):-constraint_next(X,Y,R,L,P,T).

xj(C,L,R):-solve(C,L,V),get_list(L,V,R).

get_list([],[],[]).
get_list([X|T],[A|Z],R):-A==1,get_list(T,Z,R1),R=[X|R1].
get_list([_|T],[A|Z],R):-A==0,get_list(T,Z,R).
```

La prima riga importa la libreria per poter utilizzare le funzionalità di CLP(FD) in Prolog. L'approccio utilizzato è di tipo constraint and generate, cioè prima vengono posti dei vincoli ai possibili valori che le variabili possono assumere e poi vengono generate le soluzioni che li rispettano. La regola checkpoint serve a vincolare una data variabile X a rappresentare un checkpoint in L, mentre la regola not_present serve a vincolare una variabile Y ad essere diversa da tutte le persone presenti in un dato checkpoint X, rappresentate da R e calcolata dalle regole get_people. Queste due regole verranno usate successivamente nella fase di constraint del programma. La regola solve serve a calcolare la soluzione al problema, indicata da R. Viene creata una lista delle stesse dimensioni di quella contenente i valori dei vari Y_j , con gli elementi vincolati ad assumere valore 0 o 1 (a seconda che una persona si consideri effettivamente presente in un X_j o no), poi la soluzione viene vincolata e ne viene massimizzato il numero di persone presenti. La regola ricorsiva constraints servono ad imporre i vincoli sulle soluzioni. Le regole check servono ad imporre una variabile A ad assumere valore 0 se vengono violati i vincoli sulla presenza di persone nei checkpoint adiacenti X+1 e/o X-1. In tal caso vengono posti vincoli anche su tutti i checkpoint

consecutivi con quello analizzato tramite i predicati `constraint_next` e `constraint_prev`. La regola `xj` è quella che va invocata per ottenere effettivamente la soluzione `R`, la quale indica la lista delle persone nei vari checkpoint, ottenuta attraverso la soluzione di `solve` e le regole per `get_list`.

Esperimenti

Vogliamo presentare due istanze per provare la correttezza delle nostre implementazioni. La prima è formata dall'insieme di checkpoint $C=\{2,3,4,7,8,10\}$ e dagli insiemi $Y2=\{1,3,4,9\}$, $Y3=\{1,4,10\}$, $Y4=\{1,3,4\}$, $Y7=\{3,4,5,6,7,8\}$, $Y8=\{3,5,7,8,10\}$, $Y10=\{4,5,7,9,10\}$. La soluzione ottima è data dagli insiemi $X2=X3=X4=\{1,4\}$, $X7=X8=\{3,5,7,8\}$, $X10=\{4,5,7,9,10\}$. L'istanza è rappresentata in ASP da un file contenente gli atomi che la definiscono (file *instance1_asp.lp*):

```
checkpoint(2;3;4;7;8;10).
```

```
person(1..10).
```

```
yj(2,1).
yj(2,3).
yj(2,4).
yj(2,9).
yj(3,1).
yj(3,4).
yj(3,10).
yj(4,1).
yj(4,3).
yj(4,4).
yj(7,3).
yj(7,4).
yj(7,5).
yj(7,6).
yj(7,7).
yj(7,8).
yj(8,3).
yj(8,5).
yj(8,7).
yj(8,8).
yj(8,10).
yj(10,4).
yj(10,5).
yj(10,7).
yj(10,9).
yj(10,10).
```

Per ottenere la soluzione basta lanciare da terminale il comando `clingo instance1_asp.lp solver_asp.lp`, il cui output è:

```
Answer: 20
xj(10,10) xj(10,9) xj(10,7) xj(10,5) xj(10,4) xj(8,8) xj(8,7) xj(8,5) xj(8,3) xj(7,8)
xj(7,7) xj(7,5) xj(7,3) xj(4,4) xj(4,1) xj(3,4) xj(3,1) xj(2,4) xj(2,1)
Optimization: 7
OPTIMUM FOUND

Models      : 1
Enumerated: 20
Optimum    : yes
Optimization: 7
Time       : 0.016
Prepare    : 0.000
Prepro.    : 0.000
Solving    : 0.016
```

Per quanto riguarda la soluzione in CLP invece, dopo aver richiamato il goal `consult(solver_clp.pl)`, per caricare il file contenente le definizioni sopra, basta invece richiamare il goal `xj([2,3,4,7,8,10], [[2,1],[2,3],[2,4],[2,9],[3,1],[3,4],[3,10],[4,1],[4,3],[4,4],[7,3],[7,4],[7,5],[7,6],[7,7],[7,8],[8,3],`

$[8,5],[8,7],[8,8],[8,10],[10,4],[10,5],[10,7],[10,9],[10,10]],R)$. dove il primo argomento rappresenta i checkpoint ed il secondo rappresenta i vari insiemi Y_j . L'output è il seguente:

```
R = [[2, 1], [2, 4], [3, 1], [3, 4], [4, 1], [4, 4], [7, 3], [7, 5], [7, 7], [7, 8],
      [8, 3], [8, 5], [8, 7], [8, 8], [10, 4], [10, 5], [10, 7], [10, 9], [10, 10]] ■
```

La seconda istanza invece è formata dai checkpoint $C=\{2,4,5,8,10,11\}$ e dagli insiemi $Y2=\{1,2,4,9,11\}$, $Y4=\{1,3,4,8,10\}$, $Y5=\{1,2,4,10\}$, $Y8=\{2,5,6,10\}$, $Y10=\{3,4,8,9,10,12\}$, $Y11=\{3,5,8,12\}$. La soluzione ottima è $X2=\{1,2,4,9,11\}$, $X4=X5=\{1,4,10\}$, $X8=\{2,5,6,10\}$, $X10=X11=\{3,8,12\}$. L'istanza ASP (file *instance2_asp.lp*) è la seguente:

```
checkpoint(2;4;5;8;10;11).
```

```
person(1..12).
```

```
yj(2,1).
yj(2,2).
yj(2,4).
yj(2,9).
yj(2,11).
yj(4,1).
yj(4,3).
yj(4,4).
yj(4,8).
yj(4,10).
yj(5,1).
yj(5,2).
yj(5,4).
yj(5,10).
yj(8,2).
yj(8,5).
yj(8,6).
yj(8,10).
yj(10,3).
yj(10,4).
yj(10,8).
yj(10,9).
yj(10,10).
yj(10,12).
yj(11,3).
yj(11,5).
yj(11,8).
yj(11,12).
```

Per calcolare la soluzione lanciamo da terminale il comando *clingo instance2_asp.lp solver_asp.lp*, e si ottiene:

```
Answer: 22
xj(11,12) xj(11,8) xj(11,3) xj(10,12) xj(10,8) xj(10,3) xj(8,10) xj(8,6) xj(8,5) xj(8,2) xj(5,10)
xj(5,4) xj(5,1) xj(4,10) xj(4,4) xj(4,1) xj(2,11) xj(2,9) xj(2,4) xj(2,2) xj(2,1)
Optimization: 7
OPTIMUM FOUND

Models      : 1
Enumerated: 22
Optimum    : yes
Optimization: 7
Time       : 0.000
Prepare    : 0.000
Prepro.    : 0.000
Solving    : 0.000
```

Per quanto riguarda il CLP invece, sempre dopo aver caricato le definizioni delle regole come prima, basta lanciare il goal *xj([2,4,5,8,10,11],[[2,1],[2,2],[2,4],[2,9],[2,11],[4,1],[4,3],[4,4],[4,8],*

[4,10],[5,1],[5,2],[5,4],[5,10],[8,2],[8,5],[8,6],[8,10],[10,3],[10,4],[10,8],[10,9],[10,10],[10,12],[11,3],[11,5],[11,8],[11,12]],R). L'output è il seguente:

R = [[2, 1], [2, 2], [2, 4], [2, 9], [2, 11], [4, 1], [4, 4], [4, 10], [5, 1], [5, 4], [5, 10], [8, 2], [8, 5], [8, 6], [8, 10], [10, 3], [10, 8], [10, 12], [11, 3], [11, 8], [11, 12]] ■

La terza ed ultima istanza analizzata invece è formata dall'insieme di checkpoint $C=\{2,3,5,6,7,8\}$ e dagli insiemi $Y2=\{1,2,3,5\}$, $Y3=\{1,2,3,4\}$, $Y5=\{1,2,4,6\}$, $Y6=\{1,2,4,7\}$, $Y7=\{1,2,4,8\}$, $Y8=\{1,2,4,8\}$. La soluzione ottima è $X2=X3=\{1,2,3\}$, $X5=X6=X7=X8=\{1,2,4\}$. L'istanza ASP (file *instance3_asp.lp*) è la seguente:

checkpoint(2;3;5;6;7;8).

person(1..8).

yj(2,1).
 yj(2,2).
 yj(2,3).
 yj(2,5).
 yj(3,1).
 yj(3,2).
 yj(3,3).
 yj(3,4).
 yj(5,1).
 yj(5,2).
 yj(5,4).
 yj(5,6).
 yj(6,1).
 yj(6,2).
 yj(6,4).
 yj(6,7).
 yj(7,1).
 yj(7,2).
 yj(7,4).
 yj(7,8).
 yj(8,1).
 yj(8,2).
 yj(8,4).
 yj(8,8).

Per ottenere la soluzione lanciamo da terminale il comando *clingo instance3_asp.lp solver_asp.lp*, ed otteniamo:

```

Answer: 10
xj(8,4) xj(8,2) xj(8,1) xj(7,4) xj(7,2) xj(7,1) xj(6,4) xj(6,2) xj(6,1) xj(5,4) xj(5,2) xj(5,1)
xj(3,3) xj(3,2) xj(3,1) xj(2,3) xj(2,2) xj(2,1)
Optimization: 6
OPTIMUM FOUND

Models      : 1
Enumerated: 10
Optimum    : yes
Optimization: 6
Time       : 0.000
Prepare    : 0.000
Prepro.    : 0.000
Solving    : 0.000
  
```

Il goal da invocare in CLP, sempre dopo aver caricato il file con le regole di risoluzione, è invece $xj([2,3,5,6,7,8],[[2,1],[2,2],[2,3],[2,5],[3,1],[3,2],[3,3],[3,4],[5,1],[5,2],[5,4],[5,6],[6,1],[6,2],[6,4],[6,7],[7,1],[7,2],[7,4],[7,8],[8,1],[8,2],[8,4],[8,8]],R)$. e si ottiene in output:

R = [[2, 1], [2, 2], [2, 3], [3, 1], [3, 2], [3, 3], [5, 1], [5, 2], [5, 4], [6, 1], [6, 2], [6, 4], [7, 1], [7, 2], [7, 4], [8, 1], [8, 2], [8, 4]] ■