Programmazzione Dichiarativa
Progetto: Optimal Mean Payoff Path

Matteo Capparrucci

March 2, 2016

Contents

Testo

Si consideri un grafo pesato e diretto G(V,E) i cui pesi sono sugli archi e possono essere negativi. I nodi sono partizionati in due insiemi disgiunti: $V = V_1 + V_2$. Il peso di un cammino e' la somma dei pesi dei suoi archi.

Sul grafo si puo' costruire un cammino giocando una partita tra due giocatori A_1 e A_2, in questo modo:

1. Si parte ponendo un nodo s_0 come nodo attivo.
2. Se l'attuale nodo attivo x e' in V_1 allora A_1 sceglie uno degli archi (x,y) che escono da x. Il nodo y diventa il nuovo nodo attivo. Se invece l'attuale nodo attivo x e' in V_2 allora sara' A_2 a scegliere.
3. Si ripete il passo 2) fino a quando e' possibile (cioe' si procede all'infinito oppure si giunge a un nodo senza archi uscenti).

Una strategia S per il giocatore A_1 e' una funzione da V_1 a E che per ogni nodo x in V_1 sceglie uno degli archi uscenti da x. (In pratica S guida il giocatore A_1 durante la partita).

Il grafo GS indotto dalla strategia S e' il grafo ottenuto da G eliminando tutti gli archi uscenti dai nodi V_1 tranne quelli selezionati da S. (Quindi, fissata S, il grafo GS e' la parte di G rilevante per il gioco, dato che A1 usa le scelte dettate da S).

Il problema consiste nel dire se, dato un grafo, esiste o meno una strategia S per A_1 tale che in GS non ci siano cicli di peso ¿ 0. (In tal caso A_1, usando S, vince sempre il gioco).

[Come sottoproblemi piu' semplici, si hanno: a) Dato il grafo G, dire se esiste un ciclo con peso ≤ 0 b) Dato il grafo G, dire se esiste un ciclo con peso ≥ 0]

Scrivere un programma ASP o CLP che risolve il problema.

1 Implementazione

L'implementazione del codice per la risoluzione del problema è stata eseguita utilizzando il linguaggio ASP e sfruttando il paradigma di programmazione generate and test, ovvero prima descrivendo dichiarativamente le proprietà volute nell'insieme delle soluzioni e poi eliminando tutte quelle che non soddisfano degli specifici vincoli.

Il codice è stato testato utilizzando il grounder gringo (v 4.4.0) e il solver clasp (v 3.1.1).

```
(1)     start(X,V) :- nodo(X,V),X=1.

(2)     arcoVX(X,Y) :-  arco(X,Y), nodo(X,V),start(Z,V).
(3)     nood(X) :- arcoVX(X,Y), nodo(X,V), start(Z,V).

(4)     1 {arcoSceltoVX(X, Y) : arcoVX(X,Y)} 1 :- nood(X).
(5)     arcoScelto(X,Y) :- arco(X,Y), not arcoVX(X,Y).
(6)     arcoScelto(X,Y) :- arco(X,Y), arcoSceltoVX(X,Y).

(7)     raggiunto(X) :- start(X,V).
(8)     raggiunto(Y) :- partita(X,Y), raggiunto(X).
(9)     1 {partita(X,Y) : arcoScelto(X,Y)} 1 :- raggiunto(X).

(10)    raggiungibile(X,Y) :- partita(X,Z), partita(Z,Y).
```

```
(11)    raggiungibile(X,Y) :- partita(X,Z), raggiungibile(Z,Y).

(12)    sum(N) :- N = #sum{ C,X,Y : partita(X,Y),
        raggiungibile(X,X),raggiungibile(Y,Y), costo(X,Y,C) }.
(13)    :- sum(N), N < 0.
```

Descriviamo brevemente il codice ASP sopra citato. Per prima cosa (riga 1) viene definito il nodo di partenza. Quindi, (righe 2 - 3) si individua il sotto-insieme degli archi tali che questi siano archi uscenti da un nodo dello stesso tipo di quello di partenza. Successivamente viene generato un Answer-Set per ogni strategia e viene definito il sotto-grafo GS riferito ad essa (righe 4 - 6). Quindi viene definita una condizione di *reachability* a partire dalla sorgente tale per cui vengono individuati, per ogni sotto-grafo GS, i path che portano alla costruzione di un ciclo (righe 7 - 9). Infine viene individuato, per ogni path, il sotto-insieme di questo che compone il ciclo (righe 10 - 11), e viene calcolato il costo di questo (riga 12). Come ultimo passo viene quindi aggiunto un *costraint* (riga 13) tale da eliminare i path il cui risultato sarebbe per noi di minore interesse, infatti siamo interessati a valutare l'esistenza o meno di un ciclo di costo > 0 per una data strategia e dunque i cicli < 0 possono esssere considerati trascurabili.

nota: Nello studio dei "Mean Payoff Path" ottimi si è generalmente interessati allo studio di path infiniti, ovvero di grafi contenenti cicli. Assumendo quindi di non incontrare mai dei nodi di tipo "pozzo", l'implementazione vista in precedenza di default scarta tutti quei path che non individuano un ciclo. Volendo trattare diversamente l'eventuale esistenza di nodi di tipo pozzo sarebbe quindi sufficiente aggiungere le clausole seguenti:

```
(14)    nonPozzo(X) :- nodo(X,V), arcoScelto(X,Y),X!=Y.
(15)    arcoScelto(X,X) :- nodo(X,V), not nonPozzo(X).
```

Per definire la proprietà "non essere un nodo di tipo pozzo" e conseguentemente aggiungere un arco(x,x) al sottografo GS per ogni nodo che non soddisfa la proprietà appena definita. Infine, con:

```
(16)    costo(X,X,0) :- arcoScelto(X,X).
```

Si potrebbe attribuire un peso di default per ogni arco di questo tipo in modo da poter trattare opportunamente questi path come soluzioni di "successo" o di "fallimento".

2 Esempi

Di seguito mostriamo il risultato di alcune esecuzioni del codice ASP apppena illustrato su alcuni semplici grafi di test. Nel caso dei grafi 1 e 2 l'output è stato tagliato, tramite il comando *#show predicato/arità*, ai soli campi di interesse al fine di rendere più leggibile l'output. Nell'ultimo grafo (3) invece l'output presenta il grounding completo.
In ogni grafo il nodo di partenza è evidenziato in blu.

2.1 Grafo 1

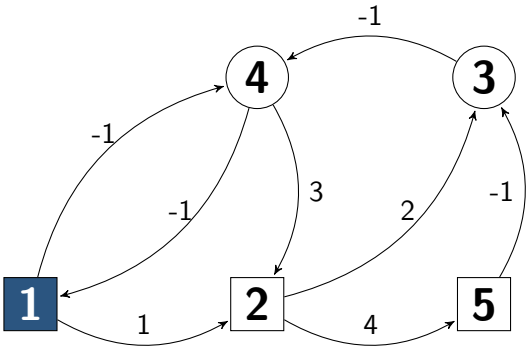

Figure 1: Grafo 1

Output:
clasp version 3.1.1
Reading from stdin
Solving...
Answer: 1
arcoSceltoVX(1,4) arcoSceltoVX(2,3) arcoSceltoVX(5,3) partita(1,4) partita(4,1) sum(-2)
Answer: 2
arcoSceltoVX(1,4) arcoSceltoVX(2,5) arcoSceltoVX(5,3) partita(1,4) partita(4,1) sum(-2)
Answer: 3
arcoSceltoVX(1,4) arcoSceltoVX(2,5) arcoSceltoVX(5,3) partita(1,4) partita(2,5) partita(4,2) partita(3,4) partita(5,3) sum(5)
Answer: 4

arcoSceltoVX(1,4) arcoSceltoVX(2,3) arcoSceltoVX(5,3) partita(1,4) partita(2,3) partita(4,2) partita(3,4) sum(4)
Answer: 5
arcoSceltoVX(1,2) arcoSceltoVX(2,5) arcoSceltoVX(5,3) partita(1,2) partita(2,5) partita(4,2) partita(3,4) partita(5,3) sum(5)
Answer: 6
arcoSceltoVX(1,2) arcoSceltoVX(2,5) arcoSceltoVX(5,3) partita(1,2) partita(2,5) partita(3,4) partita(5,3) partita(4,1) sum(2)
Answer: 7
arcoSceltoVX(1,2) arcoSceltoVX(2,3) arcoSceltoVX(5,3) partita(1,2) partita(2,3) partita(4,2) partita(3,4) sum(4)
Answer: 8
arcoSceltoVX(1,2) arcoSceltoVX(2,3) arcoSceltoVX(5,3) partita(1,2) partita(2,3) partita(3,4) partita(4,1) sum(1)
SATISFIABLE

Models : 8
Calls : 1
Time : 0.004s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time : 0.000s

2.2 Grafo 2

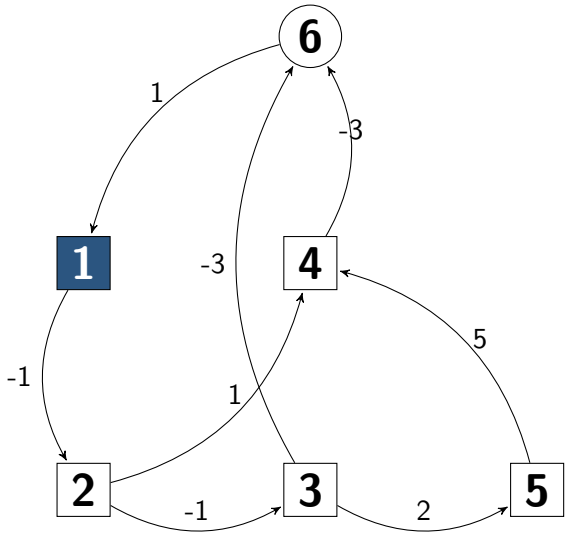

Figure 2: Grafo 2

Output:
clasp version 3.1.1 Reading from stdin
Solving...
Answer: 1
arcoSceltoVX(1,6) arcoSceltoVX(2,3) arcoSceltoVX(3,6) arcoSceltoVX(4,6) arcoSceltoVX(5,4) partita(1,6) partita(6,1) sum(3)
Answer: 2
arcoSceltoVX(1,6) arcoSceltoVX(2,3) arcoSceltoVX(3,5) arcoSceltoVX(4,6) arcoSceltoVX(5,4) partita(1,6) partita(6,1) sum(3)
Answer: 3
arcoSceltoVX(1,6) arcoSceltoVX(2,4) arcoSceltoVX(3,6) arcoSceltoVX(4,6) arcoSceltoVX(5,4) partita(1,6) partita(6,1) sum(3)
Answer: 4
arcoSceltoVX(1,6) arcoSceltoVX(2,4) arcoSceltoVX(3,5) arcoSceltoVX(4,6) arcoSceltoVX(5,4) partita(1,6) partita(6,1) sum(3)
Answer: 5
arcoSceltoVX(1,2) arcoSceltoVX(2,3) arcoSceltoVX(3,6) arcoSceltoVX(4,6) arcoSceltoVX(5,4) partita(1,2) partita(2,3) partita(3,6) partita(6,1) sum(-3)
Answer: 6

arcoSceltoVX(1,2) arcoSceltoVX(2,3) arcoSceltoVX(3,5) arcoSceltoVX(4,6) arcoSceltoVX(5,4) partita(1,2) partita(2,3) partita(3,5) partita(4,6) partita(5,4) partita(6,1) sum(4)
Answer: 7
arcoSceltoVX(1,2) arcoSceltoVX(2,4) arcoSceltoVX(3,5) arcoSceltoVX(4,6) arcoSceltoVX(5,4) partita(1,2) partita(2,4) partita(4,6) partita(6,1) sum(-1)
Answer: 8
arcoSceltoVX(1,2) arcoSceltoVX(2,4) arcoSceltoVX(3,6) arcoSceltoVX(4,6) arcoSceltoVX(5,4) partita(1,2) partita(2,4) partita(4,6) partita(6,1) sum(-1)
SATISFIABLE

Models : 8
Calls : 1
Time : 0.006s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time : 0.000s

2.3 Grafo 3

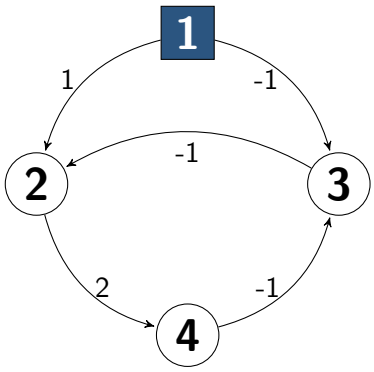

Figure 3: Grafo 3 - Rombo

Output completo:
clasp version 3.1.1
Reading from stdin
Solving...
Answer: 1
nodo(1,1) nodo(2,2) nodo(3,2) nodo(4,2) arco(1,2) arco(1,3) arco(2,4) arco(3,2) arco(4,3) costo(1,2,1) costo(2,4,2) costo(1,3,-1) costo(3,2,-1) costo(4,3,-1) start(1,1) arcoVX(1,2) arcoVX(1,3) nood(1) arcoScelto(2,4) arcoScelto(3,2) arcoScelto(4,3) arcoScelto(1,3) **arcoSceltoVX(1,3)** raggiunto(1) raggiunto(2) raggiunto(3) **partita(1,3)** raggiunto(4) **partita(2,4) partita(3,2) partita(4,3)** raggiungibile(1,4) raggiungibile(3,4) raggiungibile(1,2) raggiungibile(4,2) raggiungibile(2,3) raggiungibile(4,4) raggiungibile(2,2) raggiungibile(1,3) raggiungibile(3,3) raggiungibile(2,4) raggiungibile(3,2) raggiungibile(4,3) **sum(0)**
Answer: 2
nodo(1,1) nodo(2,2) nodo(3,2) nodo(4,2) arco(1,2) arco(1,3) arco(2,4) arco(3,2) arco(4,3) costo(1,2,1) costo(2,4,2) costo(1,3,-1) costo(3,2,-1) costo(4,3,-1) start(1,1) arcoVX(1,2) arcoVX(1,3) nood(1) arcoScelto(2,4) arcoScelto(3,2) arcoScelto(4,3) arcoScelto(1,2) **arcoSceltoVX(1,2)** raggiunto(1) raggiunto(2) **partita(1,2)** raggiunto(3) raggiunto(4) **partita(2,4) partita(3,2) partita(4,3)** raggiungibile(1,4) raggiungibile(3,4) raggiungibile(1,2) raggiungibile(4,2) raggiungibile(2,3) raggiungibile(4,4) raggiungibile(2,2) raggiungibile(1,3) raggiungibile(3,3) raggiungibile(2,4) raggiungibile(3,2) raggiungibile(4,3) **sum(0)**
SATISFIABLE

Models : 2
Calls : 1
Time : 0.001s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time : 0.000s