Calcolo della funzione di Sprague-Grundy

Paola Lorusso e Walter Mottinelli

23 aprile 2008

La funzione di Sprague-Grundy è utilizzata nella teoria dei giochi per determinare se esiste una strategia vincente per giochi combinatori imparziali.
Di seguito presentiamo l'algoritmo per calcolare la funzione di Sprague-Grundy nel caso di giochi senza ripetizione di mosse, e la sua generalizzazione al caso di giochi con ripetizione di mosse.
Implementiamo infine i due algoritmi e mostriamo i risultati della loro applicazione su grafi casuali ciclici e aciclici, che modellano le due classi di giochi sopra indicate.

1 Funzione di Sprague-Grundy

La funzione di Sprague-Grundy è utilizzata nella teoria dei giochi per determinare se esiste una strategia vincente per giochi combinatori imparziali.
Un gioco si dice **combinatorio** se soddisfa le seguenti condizioni:

- i giocatori sono due;
- l'evoluzione del gioco avviene in uno spazio degli stati finito;
- le regole del gioco determinano, in funzione dello stato attuale e del giocatore, quali sono i possibili stati futuri tra i quali il giocatore può scegliere;
- i giocatori alternano le loro mosse;
- il gioco termina quando non ci sono più mosse possibili.

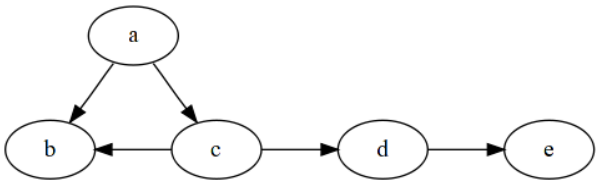
Un gioco combinatorio si dice inoltre **imparziale** se le regole non dipendono dal giocatore, altrimenti il gioco viene detto **partigiano**.
Un gioco è modellato da un grafo i cui vertici sono i possibili stati del gioco e gli archi sono le mosse legali. A partire da un qualunque stato iniziale, il gioco termina quando si raggiunge un vertice senza archi in uscita, detto anche **pozzo**. Nella modalità di

gioco normale, vince il giocatore che manda l'avversario nello stato pozzo, mentre con la regola di gioco *misère* avviene il contrario.
La funzione di Sprague-Grundy, maturata nel 1935 da Sprague e nel 1939 da Grundy e sistematizzata in [BCG82], assegna ad ogni stato del gioco un valore intero non negativo. Il valore di uno stato sarà determinato dal numero intero non negativo più piccolo esclusi i valori degli stati successivi. In base ai valori assegnati possiamo partizionare l'insieme degli stati del gioco in stati **universali** (U), che avranno valore 0, ed **esistenziali** (E). Lo stato pozzo è un esempio di stato universale.
A partire da uno stato universale, le mosse legali porteranno sempre l'avversario in uno stato esistenziale del gioco. A partire invece da uno stato esistenziale, il giocatore potrà scegliere almeno una mossa legale che porta l'avversario in uno stato universale. La strategia vincente per questo giocatore sarà individuata operando mosse che portano sempre l'avversario in uno stato universale, fino al raggiungimento del pozzo e quindi alla conclusione del gioco.

2 Algoritmo per giochi senza ripetizione di mosse

L'algoritmo presentato in 1 si applica a giochi senza ripetizione di mosse, modellati quindi da grafi aciclici, e richiede un tempo di calcolo lineare nella dimensione del grafo (non può però essere calcolata con algoritmi paralleli veloci).

2.1 Esempio di applicazione dell'algoritmo



Inizializziamo una tabella contenente, per ogni vertice v del grafo, i corrispondenti insiemi dei vertici precedenti ($Prec[v]$), dei vertici successori ($Succ[v]$) e delle loro etichette ($L\_succ[v]$). Una colonna della tabella conterrà inoltre le etichette finali dei vertici ($l(v)$, dove i vertici pozzo sono già stati etichettati con 0).
Nella colonna delle etichette dei successori, in corrispondenza dei precedenti di un vertice pozzo il valore delle etichette dei successori viene infine inizializzato a $\{0\}$.

	$Succ$	$Prec$	L_succ	l
a	$\{b, c\}$	$\{\}$	$\{0\}$	
b	$\{\}$	$\{a, c\}$	$\{\}$	0
c	$\{b, d\}$	$\{a\}$	$\{0\}$	
d	$\{e\}$	$\{c\}$	$\{0\}$	
e	$\{\}$	$\{d\}$	$\{\}$	0

Algorithm 1 Calcolo della funzione di Sprague-Grundy nel caso di giochi senza ripetizione di mosse

Require: un grafo diretto aciclico $G =< S, A >$
Ensure: il valore di Sprague-Grundy degli stati di G

```
for all y ∈ S do
  Succ[y] = {s' | (y, s') ∈ A}
end for
for all y ∈ S do
  Prec[y] = {s'' | (s'', y) ∈ A}
end for
for all y ∈ S do
  L_succ[y] = {}
end for
C = coda vuota
for all y ∈ S, y pozzo do
  l(y) = 0
  for all z ∈ Prec[y] do
    L_succ[z] = {0}
  end for
  C = accoda(y,C)
end for
while C ≠ coda vuota do
  x = primo elemento della coda C
  togli x dalla coda C
  for all v ∈ Prec[x] do
    Succ[v] = Succ[v] \ {x}
    if Succ[v] = {} then
      l(v) = MesL_succ[v]
      for all z ∈ Prec[v] do
        L_succ[z] = L_succ[z] ∪ {l(v)}
      end for
      C = accoda(v,C)
    end if
  end for
end while
```

A questo punto inseriamo nella coda C i pozzi b, e e cominciamo il ciclo principale dell'algoritmo.
Finché la coda non è vuota, estraiamo il primo elemento b e consideriamo i suoi precedenti. Per ciascuno di essi, rimuoviamo dalla tabella $Succ$ il vertice b stesso. Controlliamo quindi se i suoi precedenti hanno successori diversi da b, e poiché questo non avviene, non possiamo ancora etichettare b in alcun modo. Passiamo ora ad e, aggiorniamo $Succ$ come in precedenza, i suoi successori hanno precedenti diversi da e stesso e può quindi essere etichettato con il minimo intero non negativo esclusi i valori delle etichette dei suoi successori $L\_prec[e]$ (1).
Aggiorniamo quindi l'insieme delle etichette dei successori dei vertici precedenti d, in questo caso aggiungeremo $\{1\}$ a $L\_succ[c]$.

	$Succ$	$Prec$	L_succ	l
a	$\{c\}$	$\{\}$	$\{0\}$	
b	$\{\}$	$\{a, c\}$	$\{\}$	0
c	$\{d\}$	$\{a\}$	$\{0, 1\}$	
d	$\{\}$	$\{c\}$	$\{0\}$	1
e	$\{\}$	$\{d\}$	$\{\}$	0

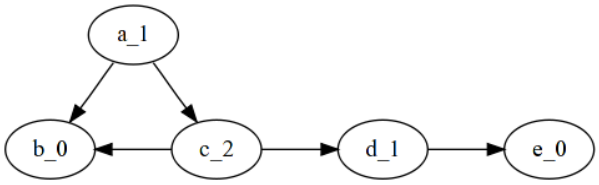
Accodiamo d in C e siccome C non è una coda vuota eseguiamo nuovamente il ciclo principale dell'algoritmo.
Estraiamo quindi d, consideriamo i suoi precedenti e ne aggiorniamo le voci nella tabella $Succ$ rimuovendo d stesso. A questo punto controlliamo se questi hanno altri successori oltre a d stesso. Poiché non avviene, il suo unico successore c viene etichettato con il minimo intero non negativo escluso i valori delle etichette dei suoi successori (2).
Come fatto in precedenza, aggiorniamo l'insieme delle etichette dei successori dei vertici precedenti c: in questo caso aggiungeremo $\{2\}$ a $L\_succ[a]$.

	$Succ$	$Prec$	L_succ	l
a	$\{c\}$	$\{\}$	$\{0, 2\}$	
b	$\{\}$	$\{a, c\}$	$\{\}$	0
c	$\{\}$	$\{a\}$	$\{0, 1\}$	2
d	$\{\}$	$\{c\}$	$\{0\}$	1
e	$\{\}$	$\{d\}$	$\{\}$	0

Accodiamo quindi c in C, lo estraiamo, aggiorniamo $Succ$ come in precedenza e controlliamo se i suoi precedenti hanno altri successori oltre a c stesso. Essendo l'unico successore di a, l'etichetta di a viene aggiornata con il minimo escluso dei valori presenti in $L\_succ[a] = 0, 2$, quindi 1. Non è necessario aggiornare la colonna L_succ perchè a non ha nessun vertice precedente. Il vertice a viene accodato in C. Il ciclo quindi viene seguito nuovamente, a viene estratto da C, ma siccome a non ha precedenti il ciclo termina e l'algoritmo pure, essendo ormai vuota la coda C.

	$Succ$	$Prec$	L_succ	l
a	$\{\}$	$\{\}$	$\{0, 2\}$	1
b	$\{\}$	$\{a, c\}$	$\{\}$	0
c	$\{\}$	$\{a\}$	$\{0, 1\}$	2
d	$\{\}$	$\{c\}$	$\{0\}$	1
e	$\{\}$	$\{d\}$	$\{\}$	0

La risultante etichettatura dei vertici è indicata nella tabella conclusiva, e nel grafico seguente dopo il simbolo "_".



3 Algoritmo per giochi con ripetizione di mosse

Consideriamo ora il caso di giochi con ripetizione di mosse. L'algoritmo precedente non garantisce l'univocità né l'esistenza della funzione di Sprague-Grundy.
La funzione però può essere generalizzata al caso di grafi ciclici, come mostrato in [Smi66] e [Fra02]. I vertici vengono partizionati in tre classi:

- alla classe U appartengono tutti i vertici in cui l'avversario del giocatore che muove da essi ha una strategia vincente;
- alla classe E appartengono tutti i vertici in cui il giocatore che muove da essi ha una strategia vincente;
- alla classe D appartengono tutti i vertici in cui nessuno dei due giocatori che muovono da essi ha una strategia vincente, ma ogni giocatore può sempre trovare una mossa non perdente.

I valori assegnati dalla funzione di Sprague-Grundy generalizzata γ sono gli interi non negativi e il valore ∞, e il partizionamento risultante è così definito:

- $U = \{u \mid \gamma(u) = 0\}$
- $D = \{u \mid \gamma(u) = \infty \wedge 0 \notin \gamma(u^+)\}$
- $E = \{u \mid 0 < \gamma(u) < \infty\} \vee \{u \mid \gamma(u) = \infty \wedge 0 \in \gamma(u^+)\}$

dove u^+ è l'insieme degli stati successivi a u.

L'algoritmo per il calcolo della funzione di Sprague-Grundy generalizzata è presentato in 2, dove S_ν è l'insieme degli stati di S non ancora etichettati con il rispettivo valore di Sprague-Grundy e u^+ è l'insieme degli stati successori di u. L'algoritmo ha complessità computazionale polinomiale $O(n(n \cdot e))$ dove n è il numero dei vertici ed e il numero degli archi.

Algorithm 2 Calcolo della funzione di Sprague-Grundy nel caso di giochi con ripetizione di mosse

Require: un grafo diretto ciclico $G =< S, A >$
Ensure: il valore di Sprague-Grundy degli stati di G

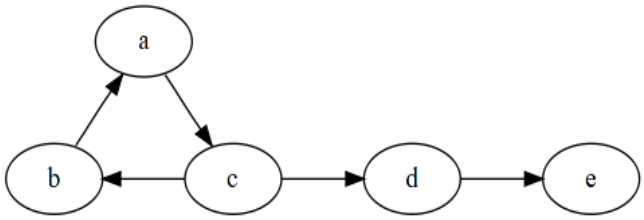
```
i = 0
m = 0
for all u ∈ S do
  l(u) = ν
end for
S_ν = S
repeat
  while ∃u ∈ S_ν : ∀v ∈ u^+ [l(v) ≠ i ∧ (l(v) = ν ∨ l(v) = ∞ ⇒ ∃z ∈ v^+ : l(z) = i)]
  do
    l(u) = i
    S_ν = S_ν \ {u}
    c(u) = m
    m = m + 1
  end while
  for all u ∈ S_ν : ∀v ∈ u^+, l(v) ≠ i do
    l(u) = ∞
    S_ν = S_ν \ {u}
  end for
  i = i + 1
until S_ν = ∅
```

3.1 Esempi di applicazione dell'algoritmo

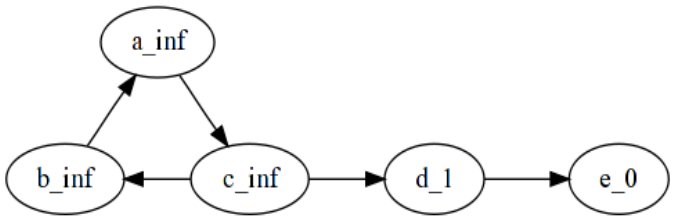
Nei seguenti esempi di applicazione dell'algoritmo, chiameremo C1 e C2 le condizioni che, se soddisfatte, portano all'etichettatura di un vertice con un valore finito, e con C3 la condizione (uguale a C1) che porta un vertice ad essere etichettato con il valore ∞:

- (C1) $\forall v \in u^+, l(v) \neq i$
- (C2) $\forall v \in u^+, l(v) \neq i \wedge (l(v) = \nu \vee l(v) = \infty \Rightarrow \exists z \in v^+ : l(z) = i)$
- (C3) $\forall v \in u^+, l(v) \neq i$

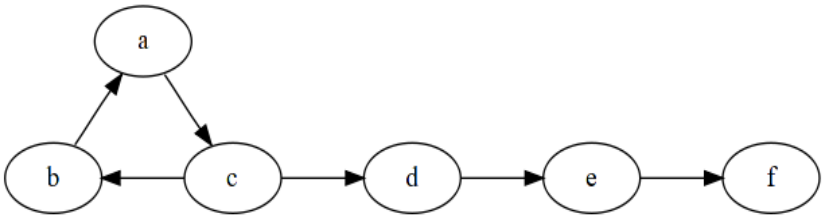
3.1.1 Caso A



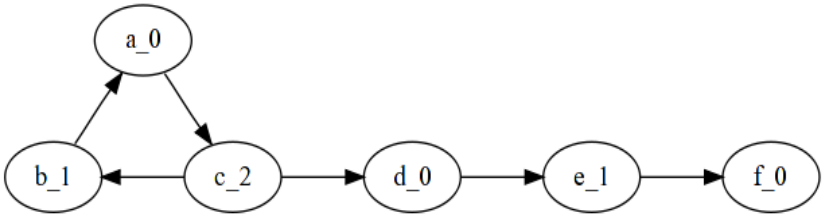
Inizializziamo i e m a 0, ed etichettiamo con ν tutti i vertici di S.
Finché tutti i vertici non sono etichettati, cerchiamo in S_ν un vertice che soddisfa le condizioni (C1) e (C2).
Osserviamo che a, b, c, d non soddisfano le condizioni richieste, mentre e può essere etichettato con 0.
Sottratto e all'insieme S_ν, controlliamo nuovamente se i restanti vertici soddisfano le condizioni (C1) e (C2): a, b, c non soddisfano la condizione (C2), mentre d non soddisfa (C1).
Controlliamo ora se i vertici in S_ν soddisfano la condizione (C3) e possono quindi essere etichettati con ∞: la risposta è positiva per a, b, c, non per d che quindi resta l'unico elemento in S_ν avendo ancora etichetta ν.
A questo punto incrementiamo i ed eseguiamo nuovamente il ciclo principale dell'algoritmo.
Il vertice d soddisfa immediatamente le condizioni (C1) e (C2) e può quindi essere etichettato con 1.
Siccome ora tutti i vertici sono stati etichettati, l'algoritmo si conclude.



3.1.2 Caso B



Inizializziamo i e m a 0, ed etichettiamo con ν tutti i vertici di S.
Finché tutti i vertici non sono etichettati, cerchiamo in S_ν un vertice che soddisfa le condizioni (C1) e (C2).
Osserviamo che a, b, c, d, e non soddisfano le condizioni richieste, mentre f può essere etichettato con 0.
Sottratto f all'insieme S_ν, controlliamo nuovamente se i restanti vertici soddisfano le condizioni (C1) e (C2): a, b, c non soddisfano (C2), d soddisfa entrambe e viene etichettato con 0, infine e non soddisfa (C1).
Controlliamo nuovamente i vertici in S_ν, a soddisfa le due condizioni e viene etichettato con 0 mentre b, c, e non soddisfano (C1). Al successivo controllo, nessuno dei restanti vertici in S_ν (b, c, e) soddisfa ancora (C1), e quindi nemmeno (C3), col risultato che nessun vertice può essere etichettato con ∞.
A questo punto incrementiamo i ed eseguiamo nuovamente il ciclo principale dell'algoritmo: b ed e soddisfano le condizioni (C1) e (C2) e quindi vengono etichettati con 1, mentre c non soddisfa prima (C1) e poi di conseguenza (C3), quindi la sua etichetta rimane ν.
Incrementiamo infine ancora i, c soddisfa le due condizioni e quindi posso etichettarlo con il valore 2.
Siccome ora tutti i vertici sono stati etichettati, l'algoritmo si conclude.



3.2 Esempi di output del programma

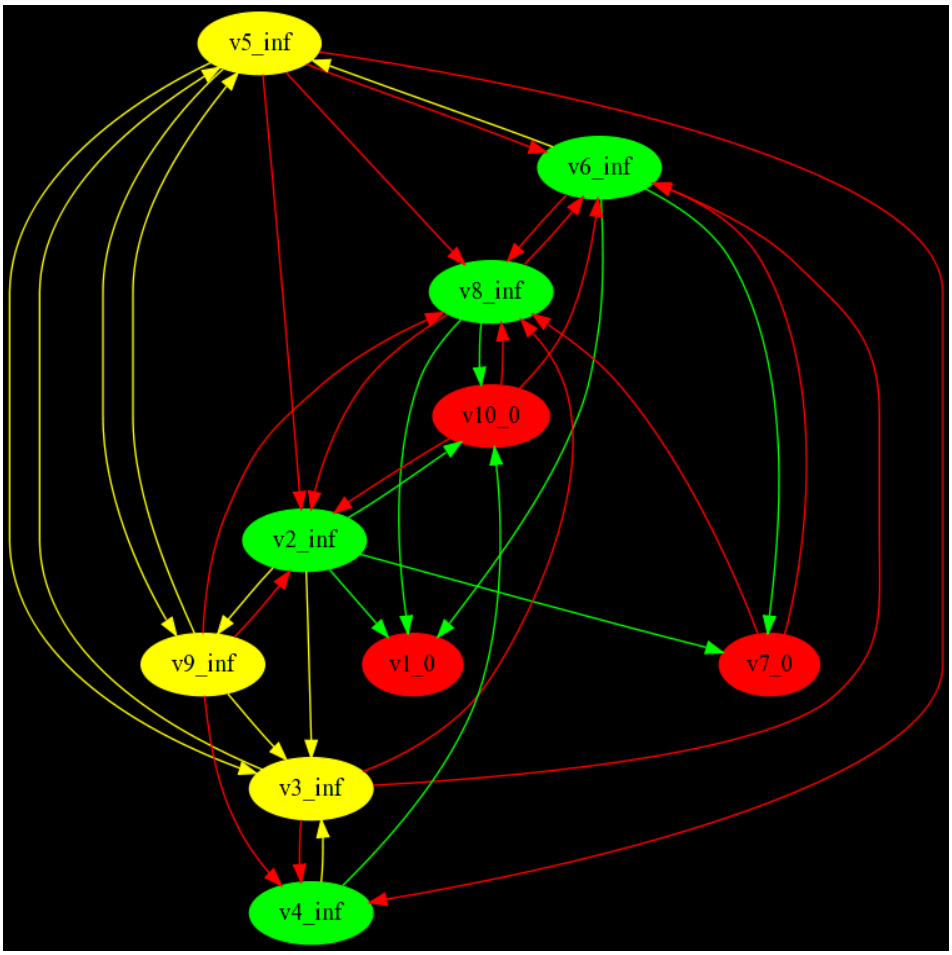
Abbiamo implementato gli algoritmi presentati in Ruby, facendo uso della libreria Ruby Graph Library (RGL) reperibile al sito http://rgl.rubyforge.org.
Il costruttore della classe `SG_solver` richiede in input un oggetto `DirectedAdjacencyGraph` e restituisce l'etichettatura dei vertici con i rispettivi valori della funzione di Sprague-Grundy. Il costruttore accetta i seguenti parametri

- il grafo `g`
- `pbar` è un valore booleano che determina la visualizzazione di una barra di avanzamento;
- `file` specifica se salvare su file una rappresentazione grafica del grafo etichettato;
- `color` specifica se la rappresentazione grafica del grafo è a colori oppure in bianco e nero;
- `circo` specifica se il grafo verrà rappresentato in forma circolare.

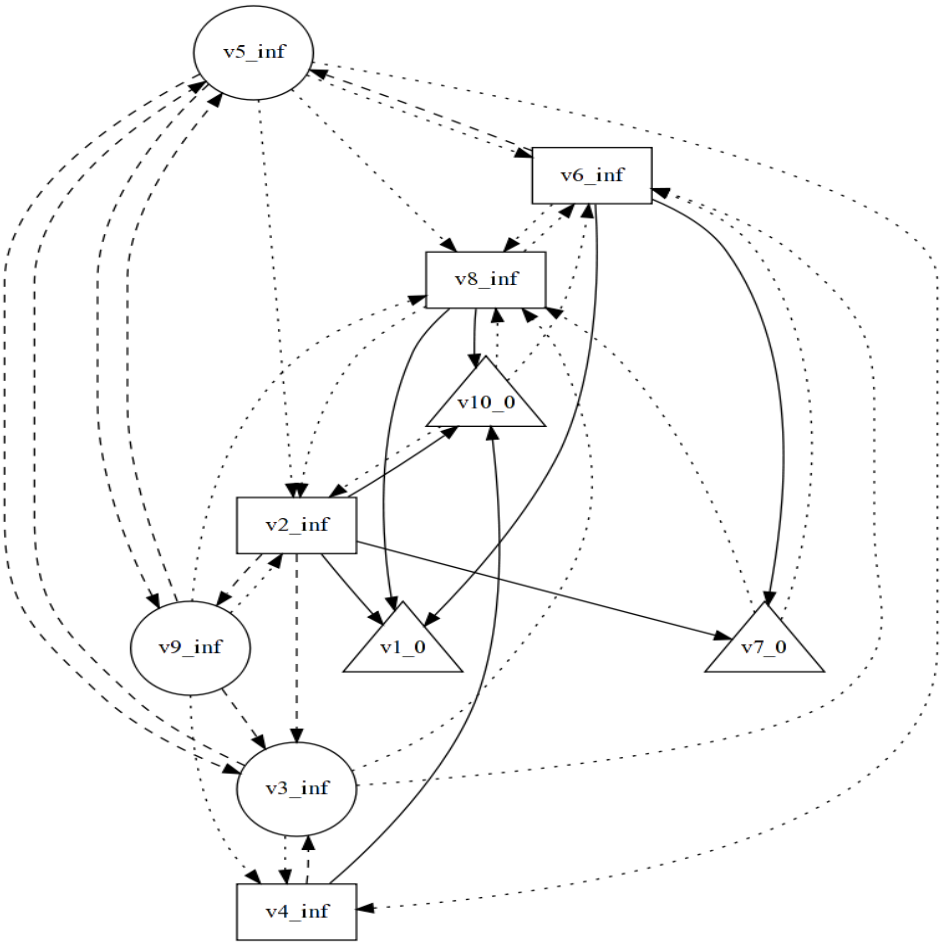
Nella rappresentazione grafica a colori del grafo, i vertici U sono contraddistinti dal colore rosso, i vertici D dal colore giallo, i vertici E dal colore verde. Nella rappresentazione in bianco e nero, i vertici sono caratterizzati non più dai colori verde, giallo e rosso, ma dalle figure geometriche (rispettivamente) rettangolo, cerchio e triangolo.
Per quanto riguarda gli archi, le mosse che portano l'avversario in uno stato E sono di colore rosso, sono infatti le mosse da evitare perché permettono all'avversario di trovarsi in uno stato esistenziale e quindi di poter seguire una strategia vincente. Per analoghi motivi, le mosse che invece portano in uno stato U sono di colore verde. Le restanti mosse, che non portano né ad una strategia vincente né ad una perdente, sono contraddistinte dal colore giallo. Nella rappresentazione in bianco e nero, gli archi sono caratterizzati non più dai colori verde, giallo e rosso, ma dal tratto (rispettivamente) continuo, tratteggiato e punteggiato.

3.2.1 Rappresentazioni di grafi etichettati con i valori di Sprague-Grundy

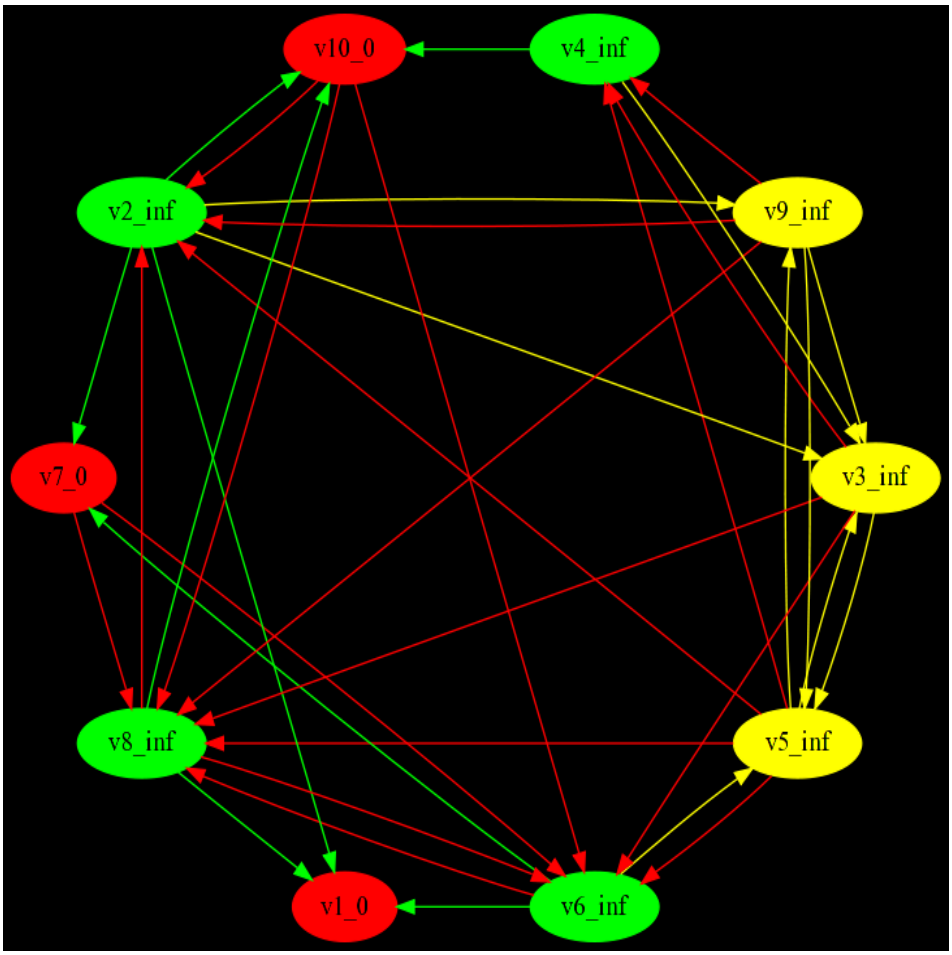
Ecco un esempio di output del programma applicato ad un grafo g, chiamando `SG_solver.new(g, :file => 'sg_graph_color.png', :color => true)`.



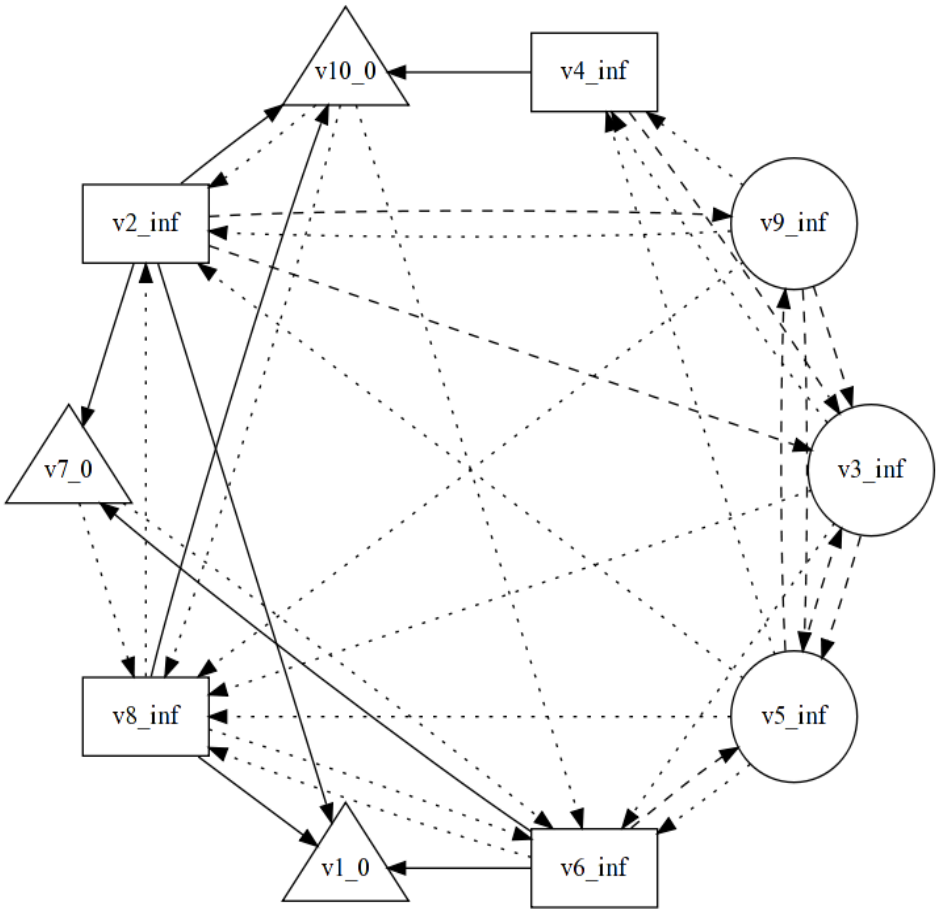
Si può ottenere la corrispondente rappresentazione grafica in bianco e nero con il comando
`SG_solver.new(g, :file => 'sg_graph_color.png', :color => false)`.



Se si desidera una rappresentazione circolare, il parametro `circo` va impostato a `true`.

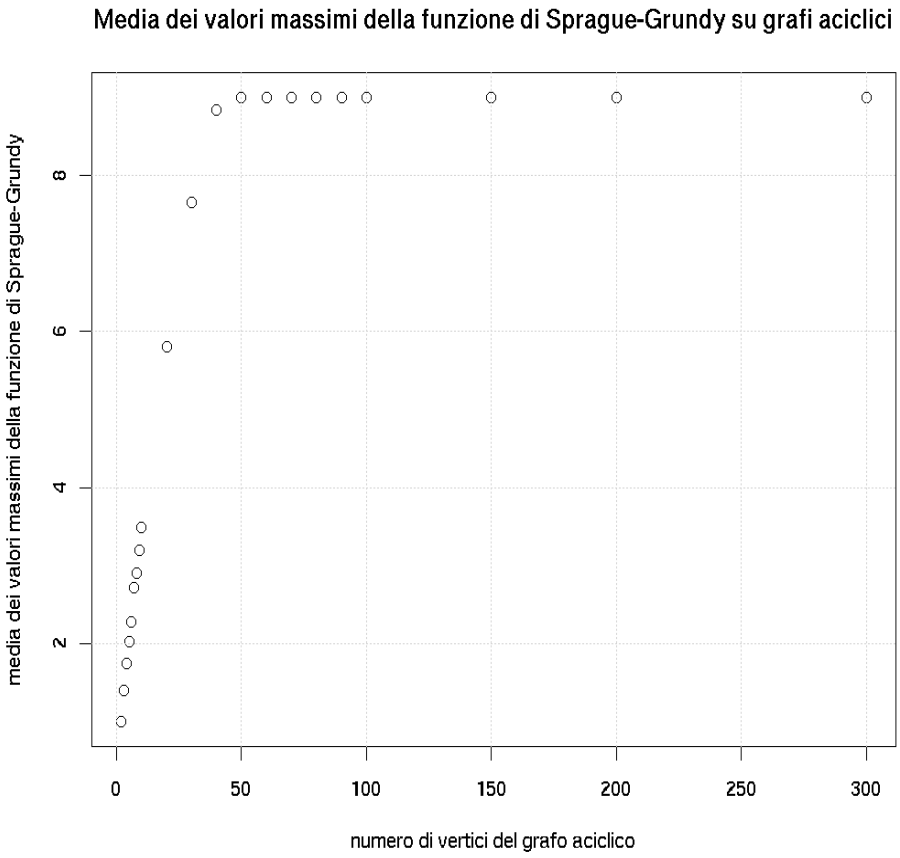


Come in precedenza, si può ottenere una rappresentazione grafica circolare anche in bianco e nero.



3.2.2 Media dei valori massimi della funzione di Sprague-Grundy su grafi aciclici

Abbiamo voluto infine valutare l'andamento del valore massimo della funzione di Sprague-Grundy al crescere del numero dei vertici n del grafo. Per diversi valori di n, abbiamo quindi generato casualmente dei grafi, sui quali abbiamo calcolato la funzione di Sprague-Grundy, e ne abbiamo fatto la media dei valori massimi. I risultati sono illustrati nella figura 3.2.2.



Come mostrato in figura, la funzione ha una crescita abbastanza rapida per valori di n minori di 50, dopo i quali la crescita sembra stabilizzarsi, o quanto meno rallentare, intorno al valore 9, almeno per $n \leq 300$.

Riferimenti bibliografici

[BCG82] Elwyn R. Berlekamp, John H. Conway, and Richard K. Guy. *Winning ways for your mathematical plays. Vol. 1: Games in general. Vol. 2: Games in particular.* London etc.: Academic Press, A Subsidiary of Harcourt Brace Javanovich, Publishers., 1982.

[Fra02] Aviezri S. Fraenkel. Two-player games on cellular automata. Nowakowski, Richard J. (ed.), More games of no chance. Cambridge: Cambridge University Press. Math. Sci. Res. Inst. Publ. 42, 279-306 (2002)., 2002.

[Smi66] C.A.B. Smith. Graphs and composite games. *J. Comb. Theory*, 1:51–81, 1966.