Generazione casuale uniforme di grafi diretti aciclici connessi

Paola Lorusso e Walter Mottinelli

22 aprile 2008

I grafi sono strumenti utili a modellare problemi di varia natura. Per studiarne statistiche e verificare ipotesi, è conveniente lavorare su grafi che siano quanto più possibile casuali.
A tal fine, alcuni autori hanno introdotto algoritmi che consentono di generare casualmente grafi diretti aciclici connessi con distribuzione uniforme, sfruttando note proprietà dei processi di Markov.
Presentiamo uno di questi algoritmi, lo implementiamo e mostriamo i risultati ottenuti.

1 Richiami sulle catene di Markov

Una **catena di Markov** è un processo stocastico a stati discreti nel quale la probabilità di transizione ad uno stato del sistema dipende unicamente dallo stato immediatamente precedente. Formalmente, chiamando $X(t)$ lo stato del sistema al tempo t,

$$P(X(t_{n+1}) = x_{n+1} \mid X(t_n) = x_n, \cdots, X(t_0) = x_0) = P(X(t_{n+1}) = x_{n+1} \mid X(t_n) = x_n)$$

Se lo spazio degli stati è finito, la probabilità di transizione da uno stato ad un altro è rappresentabile in una matrice, detta appunto **matrice di transizione**.

1.1 Proprietà di irriducibilità

Uno stato j è detto **accessibile** da uno stato differente i se, posto che siamo nello stato i, esiste una probabilità non nulla che in qualche istante futuro saremo nello stato j. Formalmente, lo stato j è accessibile dallo stato i se esiste un intero $n \geq 0$ tale che

$$P(X_n = j \mid X_0 = i) > 0$$

Uno stato i è detto **comunicante** con lo stato j se i è accessibile da j e j è accessibile da i.

Un insieme di stati C è una **classe comunicante** se ogni coppia di stati in C comunicano tra di loro, e nessuno stato in C comunica con stati non appartenenti a C.
Una catena di Markov è detta **irriducibile** se il suo spazio degli stati è una classe comunicante; significa quindi che è possibile raggiungere ogni stato a partire da ogni altro stato.

1.2 Proprietà di aperiodicità

Uno stato i ha **periodo** k se ogni ritorno allo stato i avviene in istanti multipli di k.
Se $k = 1$, allora lo stato è detto **aperiodico**.
Ogni stato in una classe comunicante deve avere esattamente lo stesso periodo.

1.3 Proprietà di ricorrenza

Uno stato i è detto **transiente** se, partendo da uno stato i, esiste probabilità non nulla di non tornare mai a i. Formalmente, sia la variabile casuale T_i il tempo del primo ritorno allo stato i (*hitting time*)

$$T_i = inf\{n : X_n = i \mid X_0 = i\}$$

Allora lo stato i è transiente se e solo se $P(T_i = \infty) > 0$.
Se lo stato i non è transiente, allora è detto **ricorrente**.
Sia M_i il tempo atteso di ritorno allo stato i

$$M_i = E[T_i]$$

Se M_i è finito, lo stato i è detto **ricorrente positivo**.

1.4 Proprietà di ergodicità

Una catena di Markov si dice **ergodica** se è aperiodica e i suoi stati sono ricorrenti positivi (in maniera equivalente, se è aperiodica, a stati finiti e irriducibile).

1.5 Stato stazionario

Se una catena di Markov è omogenea nel tempo, in modo tale che il processo può essere descritto da una singola matrice tempo-invariante p_{ij}, allora il vettore π è una **distribuzione stazionaria** se i suoi elementi π_j sommano a 1 e soddisfa la condizione

$$\pi_j = \sum_{i \in S} \pi_i p_{ij}$$

Una catena irriducibile ha una distribuzione stazionaria se e solo se tutti i suoi stati sono ricorrenti positivi. In questo caso, π è unica e vale

$$\pi_j = \frac{1}{M_j}$$

Se la catena di Markov è omogenea nel tempo, la probabilità di transizione a k-passi può essere calcolata come la k-esima potenza della matrice di transizione $\mathbf{P}$: $\mathbf{P}^k$.
La distribuzione stazionaria π è un vettore riga che soddisfa l'equazione

$$\pi = \pi\mathbf{P}$$

1.6 Distribuzione limite

Sia data una catena di Markov avente stato stazionario. Se la catena è inoltre irriducibile e aperiodica, allora per ogni i e j vale

$$\lim_{n\to\infty} p_{ij}^{(n)} = \frac{1}{M_j}$$

Se la matrice di transizione è tempo-invariante, la formula precedente può essere scritta come

$$\lim_{k\to\infty} \mathbf{P}^k = \mathbf{1}\pi$$

dove $\mathbf{1}$ è il vettore colonna con tutti gli elementi uguali a 1.
Si noti che non si fanno assunzioni sulla distribuzione iniziale: indipendentemente da essa, la catena converge alla distribuzione stazionaria.

1.7 Proprietà di reversibilità

Una catena di Markov è detta **reversibile** se esiste una distribuzione stazionaria π tale che

$$\pi_i p_{i,j} = \pi_j p_{j,i}$$

Si osservi che vale

$$\sum_i \pi_i p_{i,j} = \pi_j$$

quindi per catene di Markov reversibili, π è sempre una distribuzione stazionaria.
Una catena di Markov irriducibile e con matrice di transizione simmetrica converge alla distribuzione uniforme.

2 Metodi Monte Carlo sulle catene di Markov (MCMC)

Sfruttando le proprietà delle catene di Markov, presentiamo ora un algoritmo che genera grafi diretti aciclici (DAG) connessi con distribuzione uniforme.
Definiamo una catena di Markov M avente come stati tutti i grafi diretti aciclici sull'insieme di vertici $V = \{1, \cdots, n\}$. Lo stato al tempo t sarà identificato con X_t.
Definiamo ora la matrice di transizione secondo le regole seguenti:

1. se la coppia (i,j) corrisponde ad un arco e in X_t diretto da i a j, allora $X_{t+1} = X_t \setminus \{e\}$ (l'arco e è cancellato dal grafo associato a X_t);

2. se la coppia (i,j) non corrisponde ad un arco in X_t diretto da i a j, allora X_{t+1} è ottenuto da X_t aggiungendo questo arco, ammesso che il grafo ottenuto rimanga aciclico; altrimenti $X_{t+1} = X_t$.

Si può verificare ([MP04]) che (X_t) è una catena di Markov irriducibile e aperiodica la cui matrice di transizione è simmetrica. Quindi (X_t) ha una distribuzione limite stazionaria sull'insieme di tutti i grafi diretti aciclici costruiti sull'insieme di vertici V, e la convergenza è garantita indipendentemente dalla distribuzione iniziale.
Essendo inoltre M una catena ergodica, la distribuzione stazionaria limite a cui converge è unica, e si può verificare che è proprio la distribuzione uniforme.
Ecco l'algoritmo completo:

Algorithm 1 Generazione casuale di DAG connessi con distribuzione uniforme

Require: il numero di vertici n
Ensure: un DAG connesso generato con probabilità uniforme
  crea un semplice DAG connesso con n vertici
  loop
    estrai due vertici distinti i e j
    if esiste l'arco da i a j **then**
      cancella l'arco da i a j, a patto che il grafo resti connesso
    else
      aggiungi l'arco da i a j, a patto che il grafo resti aciclico
    end if
    altrimenti lascia il grafo inalterato
  end loop

Come detto prima, la distribuzione limite della catena di Markov è uniforme. Il problema di determinare quante transizioni siano necessarie per raggiungere una distribuzione ϵ-vicina alla distribuzione uniforme non è di facile soluzione.
In prima approssimazione si può affermare che tanto maggiore è il numero di transizioni sulla catena di Markov, tanto più la distribuzione si avvicina alla distribuzione uniforme desiderata, come mostrato sperimentalmente nel paragrafo 3.
Come anche da noi osservato sperimentalmente, l'esecuzione di un numero predeterminato di transizioni può portare ad ottenere risultati approssimati. In letteratura sono stati quindi introdotti altri metodi che permettono di campionare esattamente da una distribuzione uniforme dopo un numero di transizioni non limitato *a priori*, ma dinamicamente determinato durante l'esecuzione stessa dell'algoritmo. Si veda, ad esempio, il metodo *coupling from the past* ([PW96]). Siccome l'applicazione di questo algoritmo richiede però la conoscenza completa dello spazio degli stati, non abbiamo potuto utilizzarlo per la generazione di grafi casuali (il numero di grafi distinti aventi n vertici cresce molto rapidamente al crescere di n, come mostrato dalla sequenza http://www.research.att.com/∼njas/sequences/A082402).

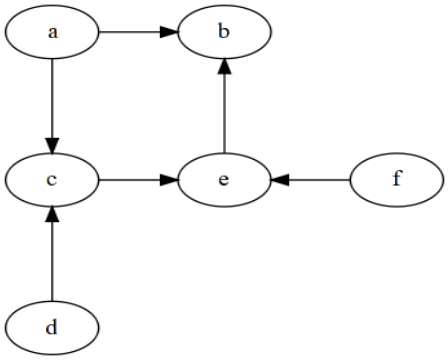
Come mostrato nell'algoritmo 1, è necessario controllare che il grafo sia aciclico e connesso dopo ogni transizione. A questo scopo, trattiamo qui di seguito gli algoritmi usati nella nostra implementazione, e ne mostriamo il funzionamento su alcuni semplici esempi.

2.1 Test di aciclicità

Per verificare l'aciclicità di un grafo, abbiamo utilizzato l'algoritmo 2.
L'algoritmo implementa una visita in ampiezza del grafo a partire dai vertici senza archi uscenti.
La complessità dell'algoritmo è $O(n+e)$ dove n è il numero dei vertici ed e è il numero degli archi.

2.1.1 Applicazione del test ad un grafo aciclico

Mostriamo con un esempio il funzionamento dell'algoritmo applicato ad un grafo aciclico.



Durante la fase di inizializzazione, la funzione *grado_entrante* associa ad ogni vertice del grafo il numero di archi entranti.

x	a	b	c	d	e	f
$grado\_entrante(x)$	0	2	2	0	2	0

Successivamente, i vertici con valori di *grado_entrante* uguali a 0 vengono inseriti in testa a *in_attesa*

$$in\_attesa = \{f, d, a\}$$

e *visitati* viene posto a 3.
Comincia quindi il ciclo principale dell'algoritmo.
Si estrae dalla testa di *in_attesa* il vertice f e si decrementa il valore di *grado_entrante* del suo successore e. Poiché $grado\_entrante(e) \neq 0$, allora *in_attesa* e *visitati* restano inalterati, rispettivamente $\{d, a\}$ e 3.

Algorithm 2 Test di aciclicità

Require: un grafo $< V, E >$, con $\mid X \mid = n$
Ensure: *true* se il grafo è aciclico, $false$ altrimenti

```
for all x ∈ V do
  grado_entrante(x) = 0
end for
for all x ∈ V do

  for all y successori di x do
    grado_entrante(y) + +
  end for
end for
in_attesa = {}
visitati = 0
for all x ∈ V do
  if grado_entrante(x) = 0 then
    in_attesa = {x} ∪ in_attesa
    visitati + +
  end if
end for
while in_attesa ≠ {} do
  x = estrai_primo(in_attesa)
  for all y successori di x do
    grado_entrante(y) − −
    if grado_entrante(y) = 0 then
      in_attesa = in_attesa ∪ {y}
      visitati + +
    end if
  end for
end while
if visitati = n then
  return true
else
  return false
end if
```

x	a	b	c	d	e	f
$grado\_entrante(x)$	0	2	2	0	1	0

Si estrae quindi d, si decrementa il valore di *grado_entrante* del suo successore c, e come nel caso precedente non si modifica né *in_attesa* ($\{a\}$) né *visitati*.

x	a	b	c	d	e	f
$grado\_entrante(x)$	0	2	1	0	1	0

Si estrae quindi a, si decrementa il valore di *grado_entrante* dei suoi successori b e c, e poiché ora il grado entrante di c è uguale a 0, viene accodato in *in_attesa* ($\{c\}$) e *visitati* viene incrementato di una unità (4).

x	a	b	c	d	e	f
$grado\_entrante(x)$	0	1	0	0	1	0

A questo punto, si estrae c da *in_attesa*, si decrementa il valore del grado entrante del suo successore e. Siccome questo diventa 0, anche e viene accodato in *in_attesa* ($\{e\}$) e *visitati* viene incrementato di una unità (5).

x	a	b	c	d	e	f
$grado\_entrante(x)$	0	1	0	0	0	0

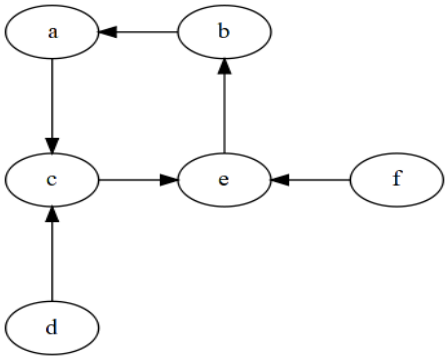
Si estrae ora e, si decrementa il grado entrante di b, siccome diventa 0 anche b viene accodato in *in_attesa* ($\{b\}$) e *visitati* viene incrementato (6).

x	a	b	c	d	e	f
$grado\_entrante(x)$	0	0	0	0	0	0

Si estrae b, il quale non ha più successori. Il ciclo principale a questo punto termina perché *in_attesa* è vuoto.
Infine, si confronta il valore di *visitati* con il numero dei vertici del grafo. Siccome in questo caso il valore è lo stesso, si può concludere che il grafo è aciclico.

2.1.2 Applicazione del test ad un grafo ciclico

Mostriamo ora il funzionamento dell'algoritmo applicato ad un grafo ciclico.



Durante la fase di inizializzazione, la funzione *grado_entrante* associa ad ogni vertice del grafo il numero di archi entranti.

x	a	b	c	d	e	f
$grado\_entrante(x)$	1	1	2	0	2	0

Successivamente, i vertici con valori di *grado_entrante* uguali a 0 vengono inseriti in testa a *in_attesa*

$$in\_attesa = \{f, d\}$$

e *visitati* viene posto a 2.
Comincia quindi il ciclo principale dell'algoritmo.
Si estrae dalla testa di *in_attesa* il vertice f, si decrementa il valore di *grado_entrante* del suo successore e. Poiché $grado\_entrante(e) \neq 0$, allora *in_attesa* e *visitati* restano inalterati, rispettivamente $\{d\}$ e 3.

x	a	b	c	d	e	f
$grado\_entrante(x)$	1	1	2	0	1	0

Si estrae quindi d, si decrementa il valore di *grado_entrante* del suo successore c, e come nel caso precedente non si modifica né *in_attesa* ($\{\}$) né *visitati*.

x	a	b	c	d	e	f
$grado\_entrante(x)$	1	1	1	0	1	0

Il ciclo principale a questo punto termina perché *in_attesa* è vuoto.
Infine, si confronta il valore di *visitati* con il numero dei vertici del grafo. Siccome in questo caso il valore è diverso, si può concludere che il grafo è ciclico.

2.2 Test di connettività

Per verificare la connettività di un grafo, abbiamo utilizzato l'algoritmo 3.

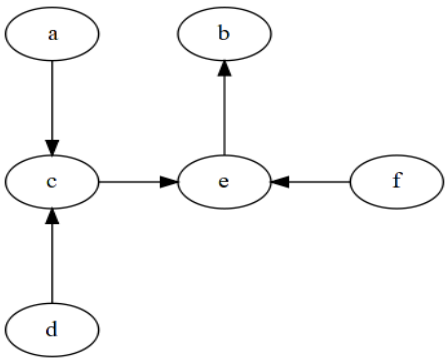
Algorithm 3 Test di connettività

Require: un grafo $< V, E >$, con $\mid X \mid = n$
Ensure: *true* se il grafo è connesso, $false$ altrimenti
  converto il grafo diretto nel suo corrispondente grafo non diretto G
  visito in ampiezza il grafo G
  conto il numero *visitati* dei vertici visitati
  if $visitati = n$ **then**
    return *true*
  else
    return $false$
  end if

L'algoritmo implementa una visita in ampiezza del grafo a partire da un vertice qualsiasi.
La complessità dell'algoritmo è $O(n+e)$ dove n è il numero dei vertici ed e è il numero degli archi.

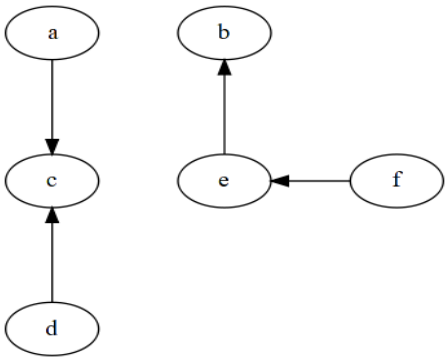
2.2.1 Applicazione del test ad un grafo connesso

Mostriamo con un esempio il funzionamento dell'algoritmo applicato ad un grafo connesso.



Inizialmente il grafo diretto viene convertito nel suo corrispondente grafo non diretto. Si visita quindi in ampiezza il grafo a partire da un vertice qualsiasi, poniamo che sia e. La visita prosegue nell'ordine c, b, f, d, a. Al termine dell'algoritmo, si verifica che il numero di vertici visitati è pari al numero di vertici del grafo, quindi si conclude che il grafo non diretto è connesso, e quindi anche il suo corrispondente grafo diretto.
Si osservi che il risultato non dipende dalla scelta del vertice iniziale.

2.2.2 Applicazione del test ad un grafo disconnesso



Anche in questo caso, l'algoritmo ha inizio con la conversione del grafo diretto nel suo corrispondente grafo non diretto.
Si visita quindi in ampiezza il grafo a partire da un vertice qualsiasi, poniamo che sia nuovamente e. La visita prosegue nell'ordine b, f, mentre a, c, b non possono essere visitati. Al termine dell'algoritmo, il numero di vertici visitati è diverso dal numero di vertici del grafo, quindi si conclude che il grafo non è connesso.
Come nell'esempio precedente, il risultato non dipende dalla scelta del vertice iniziale.

3 Risultati sperimentali

Abbiamo implementato l'algoritmo di generazione casuale uniforme di grafi diretti aciclici connessi in Ruby, facendo uso della libreria Ruby Graph Library (RGL), reperibile al sito http://rgl.rubyforge.org.
Un grafo diretto connesso casuale è generabile con la chiamata alla funzione `RGL::RandomDAG(n, iterations_number, params)`, dove

- `n` è il numero dei vertici;
- `iterations_number` è numero delle transizioni sulla catena di Markov;

e `params` è un hash di parametri opzionali:

- `acyclic` è un valore booleano che determina se il grafo sia aciclico o ciclico (default: `true`);
- `pbar` è un valore booleano che determina la visualizzazione di una barra di avanzamento (default: `true`);
- `random` specifica se usare il generatore casuale di default di Ruby oppure il generatore delle librerie Gnu Scientific Library (http://www.gnu.org/software/gsl/);
- `file` specifica se salvare su file una rappresentazione grafica del grafo.

La funzione che implementa la generazione di più grafi con le proprietà desiderate è invocata con il comando `test(graphs_n, nodes_n, transitions_n, params)`. Essa riceve in input

- il numero di grafi desiderati `graphs_n`;
- il numero di vertici `nodes_n` che caratterizzano l'insieme di grafi desiderati;
- il numero di transizioni `transitions_n` sulla catena di Markov prima di effettuare un campionamento;

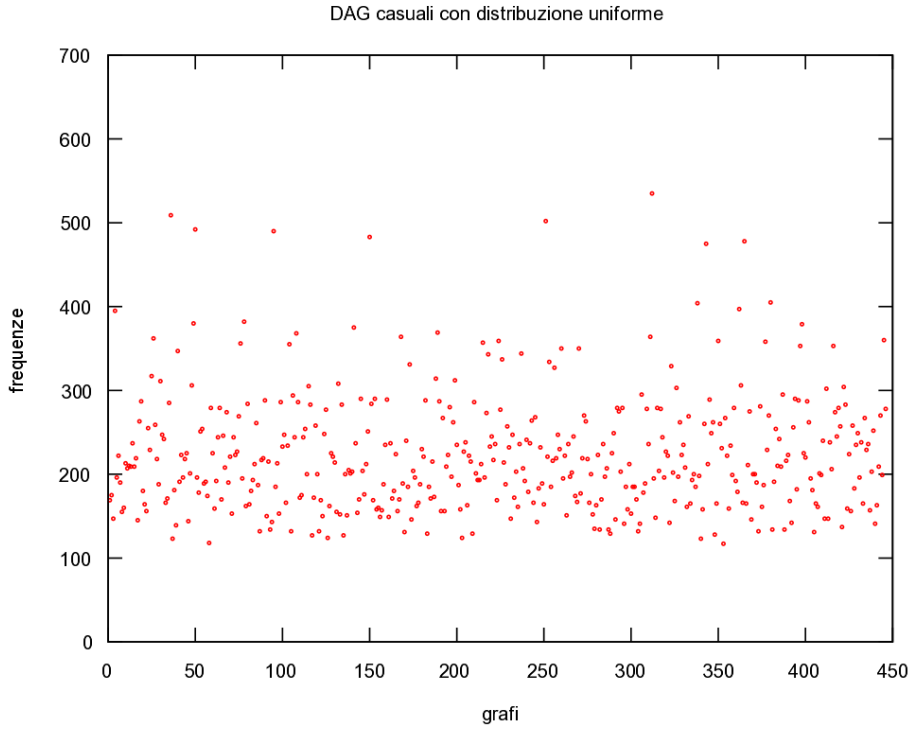
ed eventuali parametri opzionali quali

- `acyclic`, che può essere impostato a `false` nel caso in cui si desideri un grafo casuale ciclico (default: `true`);
- `pbar`, che permette la visualizzazione di una barra di avanzamento del processo di transizione sulla catena di Markov (default `true`);
- `files`, che permette di stampare su files tutti i grafi distinti generati dall'algoritmo (default `false`);
- `freq_plot`, che specifica il nome del file che conterrà il grafico del numero di grafi distinti (default `freq_plot`).

Per evidenziare che l'approssimazione della distribuzione uniforme dipende principalmente dal numero di transizioni sulla catena di Markov, abbiamo generato 100.000 grafi di 4 vertici eseguendo rispettivamente 20, 100 e 1000 transizioni. Abbiamo quindi rappresentato graficamente il numero di occorrenze di ciascun grafo distinto, così da mostrare che un numero maggiore di transizioni permette di ottenere una dispersione minore dei valori.

3.1 Generazione di grafi con 20 transizioni

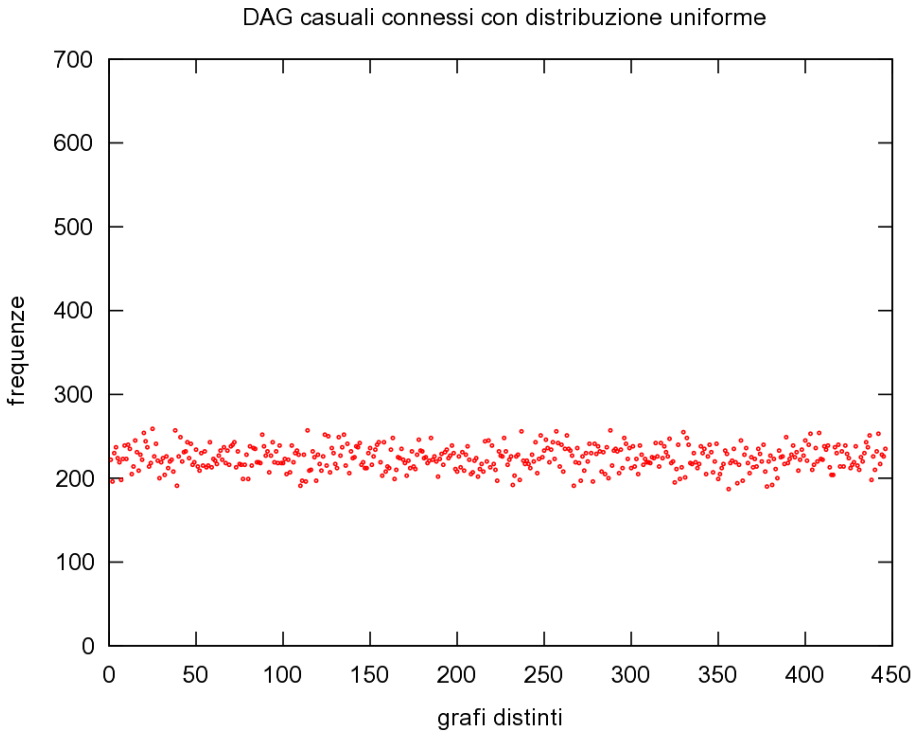
Abbiamo eseguito la generazione di 100.000 grafi distinti aventi 4 vertici utilizzando 20 transizioni sulla catena di Markov: `test(100000,4,20)`. I risultati sono mostrati nel grafico 3.1.



Osserviamo che durante l'esperimento abbiamo ottenuto 446 grafi distinti, esattamente tutti i possibili grafi diretti aciclici connessi aventi 4 vertici (la formula per il calcolo del numero di grafi diretti aciclici connessi aventi n vertici è data in [Rob73] e sul sito http://www.research.att.com/~njas/sequences/A082402).
In media abbiamo quindi ottenuto $\frac{100000}{446}$ grafi di ogni tipo, con una varianza campionaria pari a 22.1610749739784.

3.2 Generazione di grafi con 100 transizioni

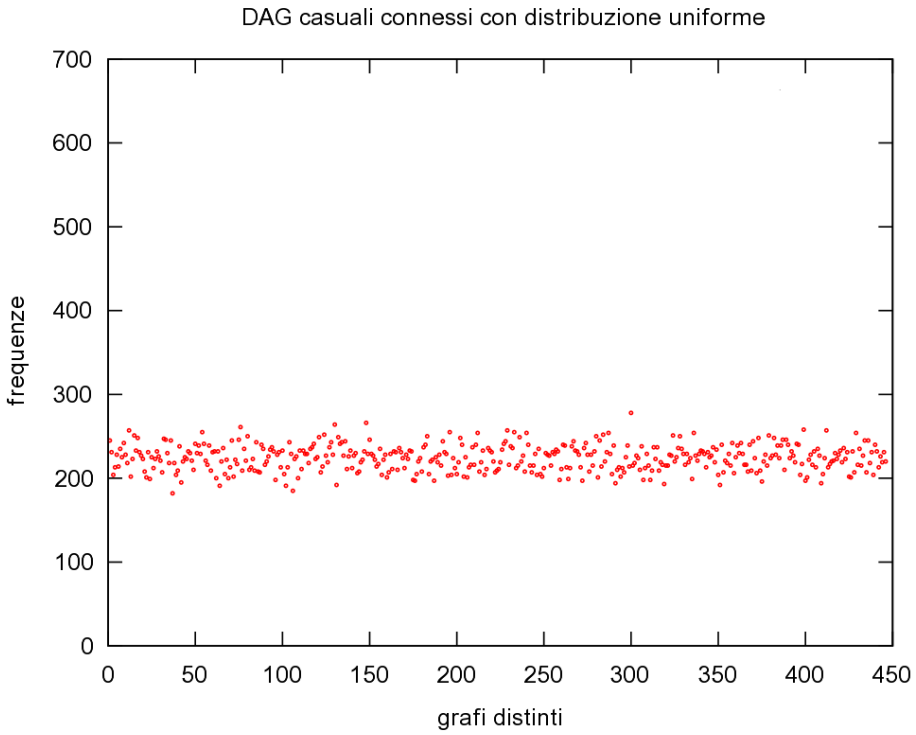
Abbiamo eseguito quindi la generazione di 100.000 grafi distinti aventi 4 vertici utilizzando 100 transizioni sulla catena di Markov: `test(100000,4,100)`. I risultati sono mostrati nel grafico 3.2.



Anche in questo esperimento abbiamo ottenuto i 446 grafi distinti attesi, ma con una varianza campionaria ridotta a 0.931768513174187. Il grafico delle frequenze mostra chiaramente la minore dispersione dei dati rispetto al caso precedente. Infatti, aumentando il numero di transizioni da 20 a 100, i risultati del campionamento mostrano che la distribuzione raggiunta dalla catena di Markov si è avvicinata sensibilmente alla distribuzione stazionaria uniforme.
Il nostro risultato è peraltro molto simile al risultato sperimentale citato nell'articolo [IC02].

3.3 Generazione di grafi con 1000 transizioni

Abbiamo quindi provato a ripetere la generazione di 100.000 grafi distinti aventi 4 vertici aumentando il numero di transizioni sulla catena di Markov a 1000: `test(100000,4,1000)`. I risultati sono mostrati nel grafico 3.3.



La varianza dei risultati non si è ridotta rispetto all'esperimento con 100 transizioni, ma è rimasta pressochè costante. La conclusione che ne abbiamo tratto è che 100 transizioni sono sufficienti perché la catena di Markov sugli stati rappresentanti grafi diretti aciclici connessi aventi 4 vertici converga alla distribuzione stazionaria, quindi eseguire ulteriori transizioni non porta ad alcun apprezzabile miglioramento.

3.4 Test non parametrici per la convergenza

Nei grafici precedenti abbiamo mostrato la convergenza della distribuzione associata alla catena di Markov campionando ripetutamente nello spazio dei DAG connessi aventi 4 vertici. Il numero di campioni da estrarre è stato stabilito in modo tale da ottenere almeno un campione per ciascun grafo distinto (446 in totale), ma perchè il test avesse rilevanza statistica il campionamento è stato effettuato un numero di volte molto più elevato (100.000 negli esempi precedenti).
Per verificare la convergenza nel caso di grafi con più vertici, è impraticabile eseguire lo stesso esperimento perchè il numero di campionamenti dovrebbe essere enorme: si consideri, ad esempio, che il numero di grafi con 20 vertici è dell'ordine di 10^{72}. Di conseguenza, per testare la convergenza è necessario utilizzare test non parametrici, ad esempio il test della somma dei ranghi di Wilcoxon (altrimenti detto test di Mann-Whitney) implementato dall'algoritmo 4. Il test restituisce il valore di probabilità

(*p-value*) che due campioni siano estratti da una stessa distribuzione.
Per campioni di numerosità maggiore di 7, si può ricorrere all'approssimazione nor-

Algorithm 4 Test di Wilcoxon

Require: due campioni A e B di numerosità n_A e n_B
Ensure: un valore di probabilità che i due campioni sono stati estratti da una stessa distribuzione
si etichettano gli elementi di ciascun campione con il nome del campione di provenienza
si uniscono i due campioni in un nuovo campione C
si ordina il campione C
si assegna agli elementi del campione C un valore (rango), pari alla posizione che occupano nell'ordinamento crescente
si somma in W_A (W_B) il rango di tutti gli elementi provenienti dal campione A (B)
si definisce $U = min\{W_A - \frac{n_A(n_A+1)}{2}, W_B - \frac{n_B(n_B+1)}{2}\}$
si calcola u, valore assunto da U nel test in corso
si consulta la tabella della distribuzione di Wilcoxon per determinare $p\_value = 2 \cdot P(U \leq u)$

male per il calcolo del *p-value* considerando la variabile casuale $Z = \frac{U-\mu_U}{\sigma_U}$, dove $\mu_U = \frac{n_A \cdot n_B}{2}$ e $\sigma_U = \sqrt{\frac{n_A n_B (n_A+n_B+1)}{12}}$.
Abbiamo quindi generato campioni di 100 grafi di 20 vertici con 20, 100, 1000, 5000 e 10000 transizioni. Abbiamo assunto che la distribuzione della catena di Markov dopo 10000 transizioni risultasse stazionaria (o quanto meno più vicina alla stazionarietà rispetto a distribuzioni ottenute con gli altri valori considerati nel test), e quindi l'abbiamo considerata come riferimento da utilizzare nel confronto con le distribuzioni ottenute con 20, 100, 1000, 5000 transizioni.
Dal confronto sono stati ottenuti i seguenti valori di probabilità:

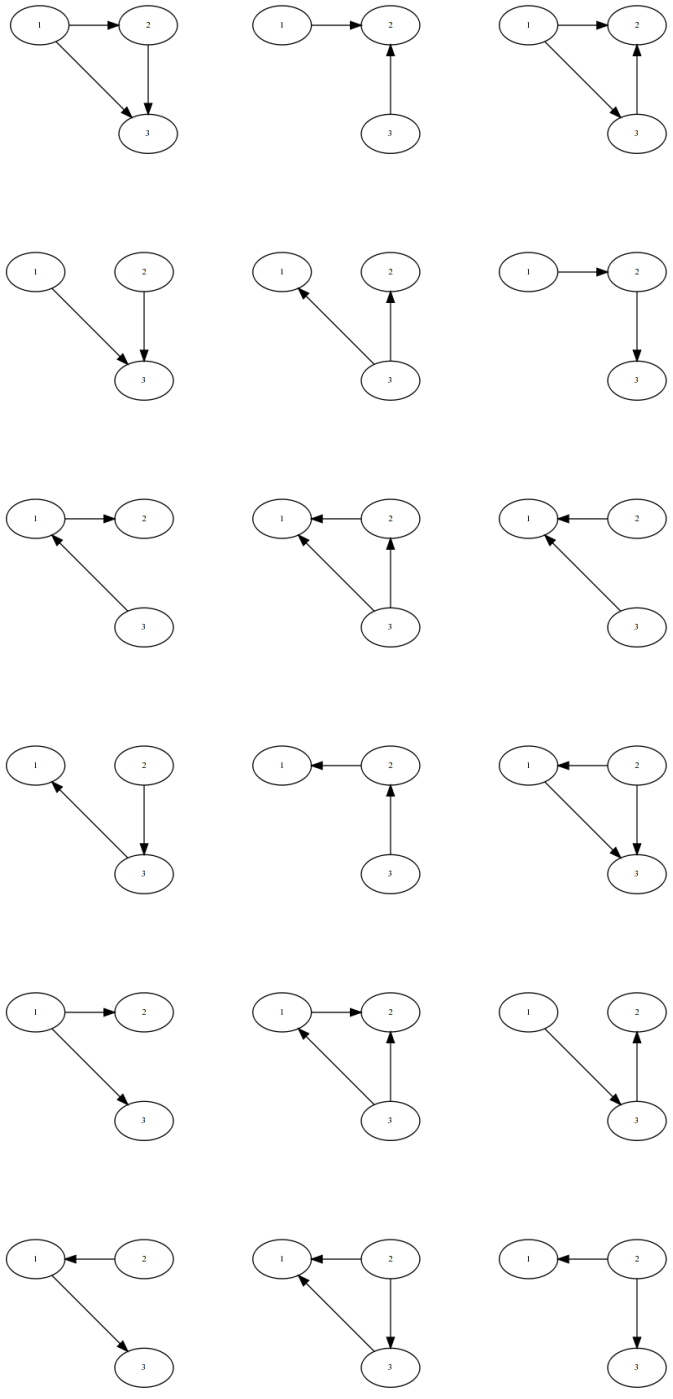
transizioni del campione A	transizioni del campione B	p-value
20	10000	2.48164354047575e-06
100	10000	0.000126870569509468
1000	10000	0.312331798334795
5000	10000	0.956157384956781

Dai dati si osserva che è altamente improbabile, come atteso, che la distribuzione di origine del campione generato con 20 o 100 transizioni sia la stessa del campione generato con 10000 transizioni. Per un numero di transizioni pari a 1000, invece la probabilità è sensibilmente più alta, e significativa dal punto di vista statistico (nel caso di test di validazione delle ipotesi, questa viene scartata per valori di $p\_value < 0.05$). Nell'ultimo caso, con 5000 transizioni, la probabilità che il campione sia stato estratto da una distribuzione identica al campione di riferimento è pari al 95%.

4 Funzionalità aggiuntive del programma

4.1 Stampa dei DAG connessi aventi un numero fissato di vertici

Il programma che implementa la generazione casuale di DAG connessi aventi distribuzione uniforme permette di stampare in files separati tutti i grafi distinti generati, impostando il parametro opzionale `files` del costruttore di `RandomDAG` a `true`.
A titolo di esempio, in figura 4.1 mostriamo i 18 DAG connessi distinti aventi 3 vertici.


1
2
3

Riferimenti bibliografici

[IC02] Jaime S. Ide and Fabio G. Cozman. Random generation of Bayesian networks. Bittencourt, Guilherme (ed.) et al., Advances in artificial intelligence. 16th Brazilian symposium on artificial intelligence, SBIA 2002, Porto de Galinhas/Recife, Brazil, November 11-14, 2002. Proceedings. Berlin: Springer. Lect. Notes Comput. Sci. 2507, 366-375 (2002)., 2002.

[MP04] G. Melançon and F. Philippe. Generating connected acyclic digraphs uniformly at random. *Information Processing Letters*, 90(4):209–213, 2004.

[PW96] James Gary Propp and David Bruce Wilson. Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Struct. Algorithms*, 9(1-2):223–252, 1996.

[Rob73] Robert W. Robinson. Counting labeled acyclic digraphs. New Direct. Theory Graphs, Proc. third Ann Arbor Conf., Univ. Michigan 1971, 239-273 (1973)., 1973.