Graphviz e Ti*k*Z

Claudio Fiandrino

Sommario

Graphviz è un potente software per disegnare grafi. Questo articolo proverà a spiegare come esportare tali grafi in figure Ti*k*Z in maniera semplice.

Abstract

Graphviz is a very powerful tool to draw graphs. This article tries to explain how to export such graphs as a Ti*k*Z picture in a very simple way.

Premessa

Nel corso dell'articolo Ubuntu 11.04 sarà il sistema operativo di riferimento. Tutti i comandi per l'installazione e la creazione del codice Ti*k*Z saranno impartiti dal terminale. Le istruzioni sono molto semplici, ma chi ha poca dimestichezza con tale strumento trova in rete alcune guide introduttive molto ben curate. A tale proposito, si riporta `http://wiki.ubuntu-it.org/AmministrazioneSistema/RigaDiComando`.

1 Introduzione

La creazione di grafi con Graphviz avviene mediante un linguaggio specifico, chiamato *dot*, la cui sintassi è veramente semplice ed intuitiva. Una breve introduzione sarà illustrata nella sezione 3. L'uso di Graphviz è largamente diffuso per analizzare, creare ed utilizzare i grafi. Un esempio pratico è la creazione di nodi di rete per simulare attacchi di pirati informatici (Zhang e Wu, 2008).

Il software che *elabora* il linguaggio dot creando il relativo codice Ti*k*Z si chiama *dot2tex*; permette di ottenere anche il codice per PGF e PSTricks. I seguenti programmi sono requisiti fondamentali per *dot2tex*:

- python, versione 2.4 o successive;
- pyparsing versione 1.4.8 (raccomandata) o successive;
- Graphviz.

Requisiti ulteriori sono i pacchetti LaTeX Preview e Ti*k*Z/PGF. Nella sezione 2 saranno illustrati i passi per installare velocemente tutti i componenti mentre nella sezione 4 sarà preso in esame il funzionamento di *dot2tex*.

2 Installazione

Python è già disponibile con l'installazione predefinita di Ubuntu (web, d); in caso contrario è sufficiente digitare da terminale:

```
sudo apt-get install python
```

Per installare *pyparsing* (web, c) e *dot2tex* (web, a) converrà usare uno strumento estremamente comodo, *easy_install*:

```
sudo apt-get install python-setuptools
```

I comandi per installare *pyparsing* e *dot2tex* sono rispettivamente:

```
sudo easy_install pyparsing
```

e

```
sudo easy_install dot2tex
```

Graphviz (web, b) necessita di una lunga lista di pacchetti di seguito citata: graphviz, graphviz-doc, graphviz-dev, libgraphviz-dev, libgraphviz-perl, libgv-lua, libgv-perl, libgv-php5, libgv-guile, libgv-python, libgv-ruby, libgv-tlc, libgv-ocaml. Possono essere installati con il solito comando:

```
sudo apt-get install <lista-pacchetti>
```

I pacchetti LaTeX Preview e Ti*k*Z/PGF sono supportati dalle distribuzioni TeX Live e MiKTeX. È consigliabile l'uso della prima citata. Una guida che illustra l'installazione su Ubuntu è (Gregorio, 2010).

3 Introduzione al linguaggio dot

Il codice 1 mostra un esempio di grafo realizzato con *dot*.

Codice 1: il primo esempio.

```
digraph G {
    1->2 [label="1/2"];
    2->3 [label="1/2"];
    3->1 [label="1/2"];
}
```

Il grafo `G` è composto da tre nodi e tre archi; dichiarato con la parola chiave `digraph`, tutto il codice che lo definisce deve essere racchiuso fra { }. È evidente l'assenza di *dichiarazioni*: i nodi e gli archi vengono creati direttamente. Una riga di codice finisce sempre con ";". Il simbolo `->` caratterizza l'arco che unisce due nodi. In questo caso l'arco ha una *direzione*, ma è possibile anche

definire archi *senza* direzione. Gli archi possono avere un'etichetta o meno; l'etichetta deve essere inserita fra parentesi quadre con la parola chiave `label`. L'esempio mostrato dal codice 1 (Il primo esempio) sarebbe notevolmente più semplice senza etichette, come mostra il codice 2 (Il primo esempio semplificato).

Codice 2: il primo esempio semplificato.

```
digraph G {
    1->2->3->1;
}
```

Per scrivere il codice del grafo è necessario creare un file con estensione `.dot`: qualsiasi editor di testo può andare bene; gli esempi qui riportati sono stati creati con *gedit*.

Ulteriori informazioni sono reperibili sul sito con la documentazione ufficiale: `http://www.graphviz.org/Documentation.php`

4 Come funziona

Per ottenere il codice di una figura non esiste un comando standard funzionale in ogni caso. A seconda dell'obbiettivo, alcuni comandi sono più *funzionali* rispetto ad altri. Ad esempio, i comandi `circo` e `neato` producono un risultato diverso a partire dallo stesso codice.

4.1 Il primo esempio

La procedura che descrive la realizzazione del primo esempio è valida anche per tutti gli altri ed è la seguente:

1. Creare una directory di nome `ex1` sul desktop.
2. Creare all'interno di tale cartella un file dal nome `ex1.dot` in cui incollare il codice mostrato nel codice 1 (Il primo esempio).
3. Aprire il terminale (click sul pulsante Applicazioni della dashboard quindi digitare `terminale` nel campo *cerca*) e scrivere:

   ```
   cd Desktop/ex1/
   ```

   per *spostarsi* all'interno della directory.
4. Creare il file `.tex` attraverso il comando:

   ```
   dot2tex --preproc ex1.dot\
   | dot2tex > ex1.tex
   ```

   Il carattere \, usato qui per esigenze tipografiche, serve comunque a far capire al terminale che il comando è stato spezzato su più righe e l'invio dopo \ non indica l'esecuzione del comando.
5. Digitare:

   ```
   pdflatex ex1.tex
   evince ex1.pdf
   ```

   I due comandi compileranno il file `.tex` creato e mostreranno il risultato, qui riportato nella figura 1, con il visualizzatore di pdf indicato.

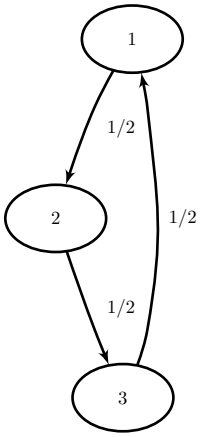


Figura 1: il primo esempio

Il documento LaTeX creato nel punto 4 è *completo*: utilizza la classe *article*, ha un proprio preambolo ed è composto da una sola pagina in cui viene inserita l'immagine.

È possibile esportare molto facilmente la figura copiando il codice, ma bisogna fare attenzione a copiare anche i pacchetti e le librerie scritti nel preambolo e necessari alla figura; in questo caso:

```
\usepackage{tikz}
\usetikzlibrary{snakes,arrows,shapes}
```

4.2 Le opzioni sul tipo di codice

Il comando *dot2tex* permette di ottenere:

- il codice PGF – opzione `fpgf`;
- il codice Ti*k*Z – opzione `ftikz`;
- il codice PSTricks – opzione `fpst`.

In generale, non specificare un'opzione fa sì che la figura sia *scritta* in PGF. Tuttavia il medesimo risultato è ottenibile digitando:

```
dot2tex -fpgf ex1.dot > ex1_pgf.tex
```

La generazione del codice Ti*k*Z avviene con:

```
dot2tex -ftikz ex1.dot > ex1_tikz.tex
```

mentre il seguente comando:

```
dot2tex -fpst ex1.dot > ex1_pst.tex
```

è necessario per ottenere un'immagine PSTricks.

I comandi riportati possono essere tranquillamente applicati al primo esempio.

4.3 Le opzioni sulle etichette dei nodi

Si consideri il codice mostrato in codice 3 (Il secondo esempio).

La procedura indicata nella sezione 4.1 implica che il file `ex2.dot` sia presente in una directory di

Codice 3: il secondo esempio.

```
digraph G {
    a_1->b_2 [label="1/2"];
    b_2->c_3 [label="1/2"];
    c_3->a_1 [label="1/2"];
}
```

nome `ex2` nel desktop. Dal codice si evince come sia facile assegnare un'etichetta ad un nodo: è sufficiente inserirla quando si crea l'arco. Un secondo metodo sarà analizzato più avanti.

Esistono tre opzioni per caratterizzare l'output:

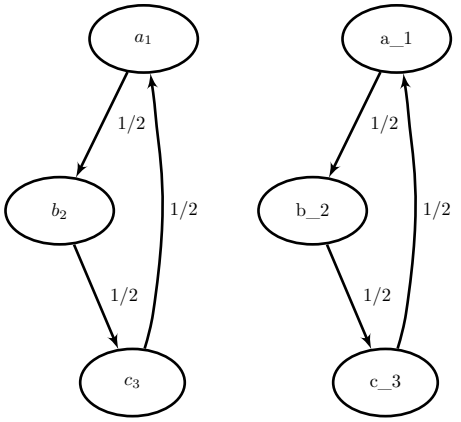
- in modalità matematica (opzione `tmath`) l'etichetta sarà inserita fra $ $;
- in modalità *verbatim* (opzione `tverbatim`) l'interprete farà sì che i caratteri speciali di LaTeX vengano riconosciuti;
- in modalità *raw* (opzione `traw`) la label non viene processata in alcun modo; se sono presenti caratteri speciali è possibile incorrere in errori.

Il comando:

```
dot2tex -tmath ex2.dot > ex2.tex
```

genera il risultato mostrato nella figura 2a, diverso da quello riportato in figura 2b ottenuto con:

```
dot2tex -tverbatim ex2.dot > ex2.tex
```



(a) label opzione *tmath* (b) label opzione *tverbatim*

Figura 2: esempi con le diverse opzioni label

Il comando:

```
dot2tex -traw ex2.dot > ex2.tex
```

fornisce invece un errore perché il carattere `_` è un carattere speciale.

Il secondo metodo che permette di specificare delle etichette per i nodi è più *pesante* di quello visto ora in quanto è necessario scrivere più righe di codice nel file `.dot` per ottenere lo stesso risultato. Ha però il grande vantaggio di essere indipendente dalle opzioni precedentemente elencate. Il motivo è molto semplice: grazie alla parola chiave `texlbl`, l'opzione da utilizzare viene definita a priori nel file `.dot`; questo è il motivo per cui è indispensabile scrivere più righe di codice.

Il codice 4 (Una variante del secondo esempio) riporta la variante del secondo esempio con le modifiche illustrate. In questo caso è sufficiente

Codice 4: una variante del secondo esempio.

```
digraph G {
    1 [texlbl="$a_1$"];
    2 [texlbl="$b_2$"];
    3 [texlbl="$c_3$"];
    1->2 [label="1/2"];
    2->3 [label="1/2"];
    3->1 [label="1/2"];
}
```

digitare

```
dot2tex ex2.dot > ex2.tex
```

per creare il file `ex2.tex` ed ottenere il risultato mostrato nella figura 2a.

4.4 Personalizzare le etichette

È possibile personalizzare il colore delle etichette attraverso la parola chiave `lblstyle`. Più in generale, è possibile introdurre tutti i comandi standard usati in TikZ. La modifica del file `ex2.dot` come riportato nel codice 5 (Il secondo esempio con label rossa) e l'immissione del comando:

Codice 5: il secondo esempio con label rossa.

```
digraph G {
  1 [texlbl="$a_1$", lblstyle="red"];
  2 [texlbl="$b_2$"];
  3 [texlbl="$c_3$"];
  1->2 [label="1/2"];
  2->3 [label="1/2"];
  3->1 [label="1/2"];
}
```

```
dot2tex ex2.dot > ex2.tex
```

genera il risultato mostrato nella figura 3.

La modifica del codice 5 riportata nel codice 6 (Il secondo esempio con comandi TikZ) illustra come è possibile introdurre altri comandi TikZ.

Come prima, il comando:

```
dot2tex ex2.dot > ex2.tex
```

genera il risultato mostrato nella figura 4.

4.5 Personalizzare i nodi

La personalizzazione dei nodi può avvenire caratterizzandone:

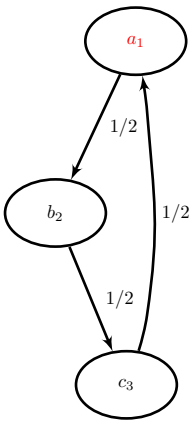


FIGURA 3: il secondo esempio con label rossa

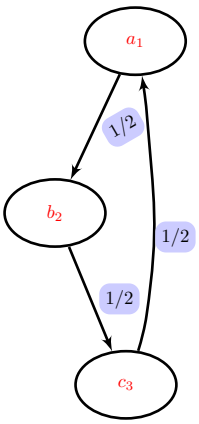


FIGURA 4: il secondo esempio con comandi Ti*k*Z

CODICE 6: il secondo esempio con comandi Ti*k*Z.

```
digraph G {
1 [texlbl="$a_1$",lblstyle="red"];
2 [texlbl="$b_2$",lblstyle="red"];
3 [texlbl="$c_3$",lblstyle="red"];
1->2 [label="1/2",
      lblstyle="rounded corners,
      fill=blue!20,rotate=30"];
2->3 [label="1/2",
      lblstyle="rounded corners,
      fill=blue!20"];
3->1 [label="1/2",
      lblstyle="rounded corners,
      fill=blue!20, below=0.1cm"];
}
```

- la forma:
  - circonferenza,
  - ellisse,
  - rettangolo;
- i colori:
  - dei bordi,
  - del riempimento;
- i font.

La parola chiave `style` e il contestuale uso dei comandi usuali di Ti*k*Z consente di personalizzare i colori e il font. La forma viene definita, invece, con la parola chiave `shape`. L'esempio `ex3.dot` illustra tali caratteristiche attraverso il codice 7 (IL TERZO ESEMPIO). C'è da notare che le impostazioni date con la parola chiave `node` diventano le impostazioni di default, ma possono essere cambiate per un singolo nodo.

È utile mostrare come la compilazione del codice 7 mediante i comandi Ti*k*Z o PGF faccia ottenere due risultati diversi. Le opzioni da utilizzare sono quelle descritte nella sottosezione 4.2:

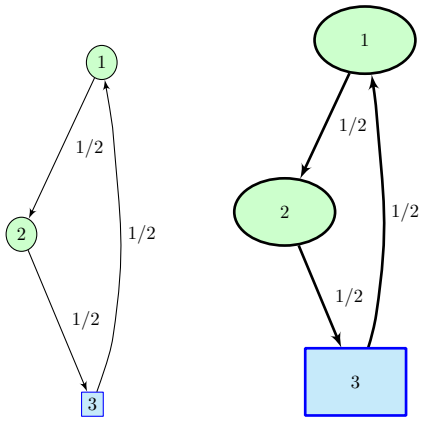
```
dot2tex -ftikz ex3.dot > ex3.tex

dot2tex ex3.dot > ex3_pgf.tex
```

CODICE 7: il terzo esempio.

```
digraph G {
    node [style="fill=green!20"];
    1->2 [label="1/2"];
    2->3 [label="1/2"];
    3->1 [label="1/2"];
    3 [shape=rectangle,
      style="fill=cyan!20,draw=blue"]
}
```

Le figure 5a e 5b mostrano tale diversità, nonostante entrambe presentino lo stesso fattore di scala. Non sono chiare le motivazioni di tale comportamento.



(a) opzione *ftikz* (b) opzione *fpgf*

FIGURA 5: esempi con le diverse opzioni di output

Si ipotizzi sia necessario caratterizzare diversi tipi di nodi in un grafo, ad esempio una rete di sensori in un'area geografica che monitori la temperatura oppure una rete di antenne accese e spente. *Colorando* in modo diverso i nodi sarebbe possibile visualizzare immediatamente quali antenne sono attive oppure quali sensori misurano un aumento della temperatura.

Risolvere questo problema caratterizzando i nodi come fatto in precedenza sarebbe poco pratico.

L'impostazione di default unica implicherebbe dover agire manualmente per modificare i nodi del secondo tipo. Sulla documentazione online questo problema non è affrontato, anche se è opinione dell'autore che rivesta notevole importanza. La soluzione proposta utilizza la parola chiave `d2tfigpreamble`; il codice 8 (Il quarto esempio) riporta un esempio in cui viene illustrato tale metodo.

Codice 8: il quarto esempio.

```
digraph G {
 d2tfigpreamble="
 \tikzstyle{cold}=\ [draw=blue!50,
 very thick,fill=blue!20],
 \tikzstyle{hot}=\ [draw=red!50,
 very thick,fill=red!20]";
 A [style="cold"];
 B [style="cold"];
 C [style="hot"];
 D [style="hot"];
 A->B->D;
 A->C->D;
 D->C;
 D->B->A;
 B->B [topath="loop left"];
 C->B;
}
```

È molto importante notare la sintassi: entrambe le definizioni degli stati sono racchiuse fra *doppi apici*; è necessario inserire un *backslash* prima della parola chiave `tikzstyle` e prima di definire le caratteristiche del nodo fra [ ]. Inoltre, come succede per Ti*k*Z, i vari comandi devono essere separati da una *virgola*.

Naturalmente è possibile definire anche più di due *stati* con questo metodo.

Rimane da analizzare come comandi diversi di Graphviz generano risultati diversi partendo dal medesimo esempio. Il comando:

```
circo -Txdot ex4.dot |\
dot2tex -ftikz > ex4.tex
```

genera il grafo riportato nella figura 6.

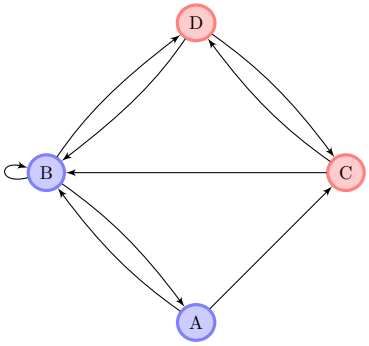


Figura 6: il quarto esempio comando *circo*

Utilizzando invece il comando:

```
circo -Txdot ex4.dot |\
dot2tex -ftikz --styleonly > ex4.tex
```

e seguendo sempre la procedura illustrata nella sottosezione 4.1 il risultato ottenuto, riportato nella figura 7, sarà diverso.

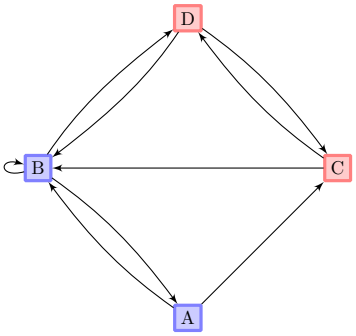


Figura 7: il quarto esempio opzione *styleonly*

È facile intuire che il motivo di tale differenza è dovuto all'opzione `styleonly`. Se l'opzione è assente, il codice Ti*k*Z che definisce un nodo è:

```
\node (A) at (108.21bp,19.5bp)
  [draw,ellipse,cold] {A};
```

mentre, quando è attiva il codice è:

```
\node (A) at (108.21bp,19.5bp)
  [cold] {A};
```

In ultimo, il comando:

```
neato -Txdot ex4.dot |\
dot2tex -ftikz --styleonly > ex4.tex
```

genera il grafo visibile nella figura 8.

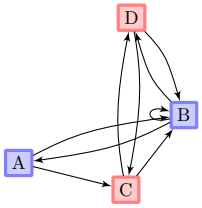


Figura 8: il quarto esempio comando *neato*

Il confronto tra le figure 6 e 8 fa dedurre che il comando `circo` è preferibile con topologie circolari perché utilizza un algoritmo che *posiziona* i nodi mentre `neato` no. Il manuale di Graphvix conferma l'osservazione.

4.6 Personalizzare gli archi

Esistono due metodi per personalizzare gli archi: mediante la sintassi `edge[style="..."]` oppure attraverso la parola chiave `topath`.

Il codice 9 (Il quinto esempio) analizza il primo approccio.

Si noti come la sintassi che caratterizza l'arco deve essere inserita prima del codice di definizione. Il risultato grafico mostrato in figura 9 è stato ottenuto con il comando:

Codice 9: il quinto esempio.

```
digraph G {
 d2tfigpreamble="
 \tikzstyle{cold}=\ [draw=blue!50,
 very thick,fill=blue!20],
 \tikzstyle{hot}=\ [draw=red!50,
 very thick,fill=red!20]";
 A [style="cold"];
 B [style="cold"];
 C [style="hot"];
 D [style="hot"];
 edge [style="snake=zigzag"];
 A->B->D->C->A;
 edge [style="snake=snake"];
 A->D;
 C->B;
}
```

```
circo -Txdot ex5.dot |\
dot2tex -ftikz -s > ex5.tex
```

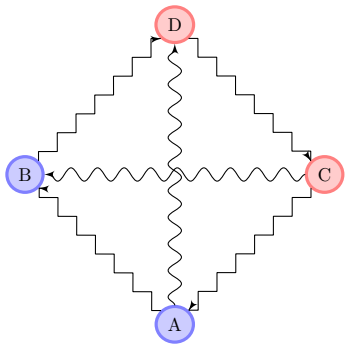


Figura 9: il quinto esempio

L'opzione `-s` deve essere specificata per poter riconoscere i percorsi di tipo *snake*. La libreria Ti*k*Z richiesta dal comando

```
\usetikzlibrary{snakes}
```

è *snakes*, una libreria obsoleta recentemente sostituita da *decorations*. Questo è il motivo per cui le forme selezionabili sono poche. Essere vincolati a una libreria obsoleta non permette neanche l'uso di opzioni estremamente interessanti come *pre/-post length*. Queste permettono di personalizzare il punto di inizio e fine del cambiamento di forma dell'arco. Ovviamente è possibile editare il codice Ti*k*Z in un secondo momento, ad esempio inserendo la libreria *decorations* e il codice relativo alla forma dell'arco. È sufficiente individuare la parte di codice giusta:

```
\draw [->,snake=zigzag] (C) -- (A);
\draw [->,snake=snake] (A) -- (D);
```

Il codice 10 (Il sesto esempio) permette invece l'analisi del metodo che fa uso della parola chiave `topath`.

Questo esempio è diverso dal precedente poiché mostra una proprietà diversa: non viene cambiata la forma dell'arco ma l'angolazione di partenza ed arrivo dell'arco dai/nei nodi. Entrambi gli approcci permettono l'uso dei classici comandi Ti*k*Z essendo, pertanto, equivalenti. È dunque possibile inserirli contemporaneamente nello stesso codice.

Codice 10: il sesto esempio.

```
digraph G {
 d2tfigpreamble="
 \tikzstyle{cold}=\ [draw=blue!50,
 very thick,fill=blue!20],
 \tikzstyle{hot}=\ [draw=red!50,
 very thick,fill=red!20]";
 A [style="cold"];
 B [style="cold"];
 C [style="hot"];
 D [style="hot"];
 A->B->D->C->A[topath="bend left=20"];
 A->C->D->B->A[topath="bend left=20"];
 B->B [topath="loop left"];
}
```

Il comando:

```
circo -Txdot ex6.dot |\
dot2tex -ftikz > ex6.tex
```

permette di ottenere il risultato riportato nella figura 10.

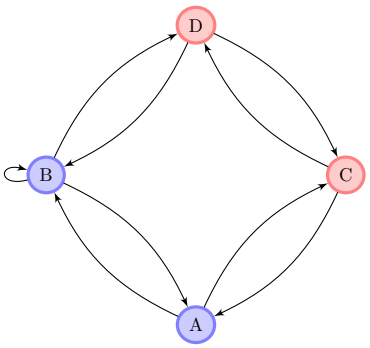


Figura 10: il sesto esempio

5 Considerazioni finali

Questa sezione cercherà di rispondere alla domanda "perché non scrivere il codice del grafo direttamente con Ti*k*Z?"

Non esiste una risposta univoca. I vantaggi del metodo illustrato sono evidenti quando occorra importare in un documento dei grafi con un certo numero di nodi ed archi. Se invece il grafo ha due o tre nodi forse è più conveniente scrivere direttamente il codice Ti*k*Z. La figura 11 è un esempio di questo tipo e raffigura una semplice catena di Markov. Il sorgente è riportato in codice 11 (grafo composto con Ti*k*Z).

C'è da notare che, a differenza del codice *dot*, debba essere indicata la posizione dei nodi e la sintassi del codice sia leggermente più complicata. Si può quindi affermare che il linguaggio *dot*:

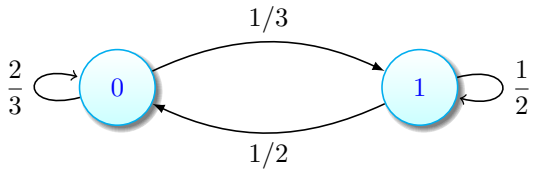


Figura 11: un esempio di grafo composto direttamente con TikZ

Codice 11: un grafo composto direttamente con TikZ.

```
\begin{tikzpicture}[-latex,auto,
  node distance=4cm,on grid,semithick,
  state/.style={circle, draw,
  top color=white,
  bottom color=cyan!20, cyan,
  circular drop shadow,text=blue,
  minimum width=1cm}]
% Nodes
\node[state] (A) {$0$};
\node[state] (B) [ right=of A] {$1$};
% Edges
\path (A) edge [loop left]
  node{$\dfrac{2}{3}$} (A);
\path (B) edge [loop right]
  node{$\dfrac{1}{2}$} (B);
\path (B) edge [bend left=25]
  node[below]{$1/2$} (A);
\path (A) edge [bend right=-25 ]
  node[above]{$1/3$} (B);
\end{tikzpicture}
```

- è più intuitivo;
- non richieda di posizionare manualmente i nodi perché lo fa autonomamente in base al comando usato (quindi all'algoritmo opportuno);
- permetta di ottenere la stessa qualità di TikZ grazie alla possibilità di personalizzare gli *stati* attraverso i preamboli.

6 Conclusioni

Questo articolo ha analizzato un metodo per esportare grafi creati con Graphviz in immagini prodotte con codice TikZ, PGF e PSTricks.

Dopo aver illustrato come installare i programmi necessari, sono stati presentati numerosi esempi (codice e risultato grafico) per discutere i punti di forza e di debolezza del metodo.

L'articolo non è esaustivo avendo discusso solo gli aspetti di base dei comandi introdotti e non avendo preso in considerazione il pacchetto *dot2texi*. In teoria, grazie all'ambiente dot2tex, l'uso del pacchetto dovrebbe essere preferibile al metodo illustrato in questo articolo: il codice dot viene inserito nell'ambiente citato e produce automaticamente la figura. Le compilazioni fatte sugli esempi tratti dal manuale di *dot2texi* danno sempre un errore in corrispondenza della linea di apertura dell'ambiente, errore che comunque non pregiudica l'ottenimento del grafo finale, seppur leggermente diverso da quanto riportato nel citato manuale.

Riferimenti bibliografici

(a). «dot2tex». URL `http://www.fauskes.net/code/dot2tex/`.

(b). «Graphviz». URL `http://www.graphviz.org/Download.php`.

(c). «Pyparsing». URL `http://pyparsing.wikispaces.com/`.

(d). «Python». URL `http://www.python.org/`.

Gregorio, E. (2010). «Installare TEX live 2010 su ubuntu». *ArsTEXnica*, (10), pp. 7–13. URL `http://www.guit.sssup.it/arstexnica/`.

Zhang, T. e Wu, C. (2008). «Network security analysis based on security status space». *Web-Age Information Management*. URL `http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4597065`.

▷ Claudio Fiandrino
http://claudiofiandrino.altervista.org
`claudio dot fiandrino at gmail dot com`