Visualización de grafos

Mario Alberto Gutiérrez Carrales
9273

26 de febrero de 2019

Introducción

Este reporte está enfocado en la visualización de grafos mediante distintos algoritmos de acomodo que brinda `python` [1] en sus diferentes módulos.

Cuando recién se comienzan a usar los grafos que la librería Networkx ofrece, el usuario puede enfrentarse a situaciones como tener que posicionar los nodos para visualizar la representación de la red, justo como se observó en la tarea 1 [2]. Ante esta problemática y otras que pudieran existir, se proponen los algoritmos de acomodo (inglés: layout) que su objetivo es representar un grafo con ciertas características de la mejor manera visualmente posible, cabe destacar que en este caso se mostrará un algoritmo para cada ejemplo realizado en la práctica anterior.

Antes de comenzar con los ejemplos resulta interesante mencionar los aspectos computacionales que pudieran ser más relevantes o retadores en la codificación.

Aspectos computacionales

Se puede comenzar por describir las librerías que se utilizan para obtener exitosamente las representaciones de cada uno de los grafos, las cuales son **NetworkX** [3], **Matplotlib** [4] y **PyGraphviz**[5].

La primera proporciona los elementos que un grafo puede tener, como su estructura tanto de nodos como aristas. La segunda proporciona visualizaciones tanto en la consola como para guardar imágenes en el formato *eps* que fue el que se utilizó para la obtención de las figuras. Por último, cabe mencionar que la primer librería mencionada en este apartado a pesar de ser muy extensa, puede complementarse con algunas otras en el manejo de los grafos y especialmente en los algoritmos de acomodo, esta es una de las principales funciones de la tercer librería.

En cualquier lenguaje de programación se puede ver que hay partes repetitivas tales como librerías, impresiones, declaración de variables, etcétera y en python no es la excepción, así que con el objetivo de evitar fragmentos de códigos repetitivos se muestran inicialmente las partes que pueden presentarse con mayor frecuencia.

Código 1: *Librerías usadas para la visualización de grafos*

```
1 import networkx as nx
2 import matplotlib.pyplot as plt
3 import pygraphviz as pgv
```

Código 2: *Quitar ejes y guardar el grafo en una imagen en formato eps*

```
1 plt.axis('off')
2 plt.savefig("Ejemplo1.eps")
3 plt.show()
```

En el código 1 se muestran las librerías utilizadas y en el 2 como guardar una imagen en formato eps.

Otro aspecto que más que repetitivo es tedioso, es cuando se pretende guardar una imagen, pero debido a las dimensiones del grafo se cortan los márgenes y se pierden información visual. Una manera de evitar esta situación está dada en el código 3.

Código 3: *Agrandar márgenes*

```
plot_margin = 0.05

x0, x1, y0, y1 = plt.axis()
plt.axis((x0 - plot_margin,
          x1 + plot_margin,
          y0 - plot_margin,
          y1 + plot_margin))
```

Extraído de stackoverflow.com

En diversas versiones de python se presentan dificultades para trabajar con la librería pygraphviz como solución alterna se propone el uso de la versión web de colab de google, disponible en: `https://colab.research.google.com/notebooks/welcome.ipynb`

Para tratar de brindar la mayor uniformidad posible a las representaciones visuales, se utilizan arcos **negros** para indicar una arista simple, arcos **rojos** a aquellos aristas que contienen múltiples aristas, nodos **azules** que indican un nodo simple, y nodo **verde** para aquellos que tienen un lazo.

Se puede observar inicialmente que cada grafo puede ser representado visualmente de manera aceptable por más de un algoritmo, la elección entre algoritmo - grafo fue de acuerdo simplemente con que el dibujo realmente representara la situación de la que se está tratando.

1. Spectral

Este grafo representa las eliminatorias de un torneo de futbol. Se constan de 4 semifinalistas, de los cuales solamente 2 pasan a la siguiente fase y uno de ellos es quien queda campeón. El algoritmo que brinda una mejor representacion es el spectral layout, se puede programar como se muestra en el código 4 y el resultado de la visualización se muestra en la figura 1.

Código 4: *Spectral layout*

```
G = nx.Graph()

Conexion=[('A', 'B'),('A','C'),('B','D'),('B','E'),('C','F'),('C','G')]
G.add_edges_from(Conexion)

labels={'A':'C','B':'F1','C':'F2','D':'SF1','E':'SF2','F':'SF4','G':'SF3'}

pos = nx.spectral_layout(G)

nx.draw_networkx_nodes(G,pos,node_color='b',alpha=1,node_size=500)
nx.draw_networkx_edges(G,pos,edge_color='k',alpha=1,width=3)
nx.draw_networkx_labels(G,pos,labels,font_size=10,font_color='w')
```

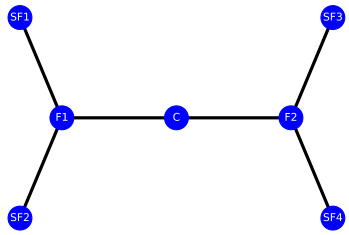

Figura 1: *Semifinales de torneo de futbol*

Se observa que la representación visual es bastante buena debido a la característica de árbol que esta situacion exige, otro algoritmo que pudo haber sido considerado es el bipartite layout.

2. Bipartite

Una de las aplicaciones más utilizadas hoy en dia es la representación de usuarios de una red social como Facebook, en ocasiones combiene saber dicha relación por género.

Código 5: *Bipartite layout*

```
G = nx.Graph()
G.add_nodes_from(range(6))
G.add_edges_from([(0,1),(1,2),(3,4),(1,5),(2,4),(4,5)])

pos = nx.bipartite_layout(G, {1,4})

labels={0:'Ana',1:'Juan',2:'Paty',3:'Dany',4:'Jose',5:'Zoe'}

nx.draw_networkx_nodes(G,pos,node_color='b',alpha=1,node_size
    =600)
nx.draw_networkx_edges(G,pos,edge_color='k',alpha=1,width=3)
nx.draw_networkx_labels(G,pos,labels,font_size=10,font_color='w'
    )
```

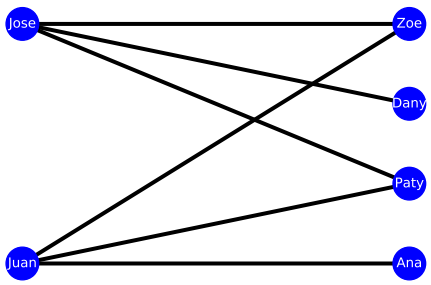

Figura 2: *Relación de amistad por género*

Como la situación está dada por género esto índica directamente a una partición del grafo en 2 secciones lo cual da la intuición que no hay mejor representación que la que se utilizó.

3. Random

Este ejemplo se basa en como están construidos ciertos cruces de avenidas en diferentes ciudades, de tal manera que las avenidas son perpendiculares entre sí y en medio se encuentra un retorno, que representa un lazo en el grafo, y su objetivo es cambiar la orientación en la que se dirige el automovilista. Observar figura 3.

Código 6: *Random layout*

```
G = nx.Graph()

Conexion=[('A', 'B'),('A','C'),('B','D'),('B','E')]
G.add_edges_from(Conexion)

labels={'A':'Esq\n #1','B':'Retorno','C':'Esq\n #2','D':'Esq\n #3','E':'Esq\n #4'}

pos = nx.random_layout(G)

nx.draw_networkx_nodes(G,pos,node_size=1000,nodelist=['B'],
    node_color='g')
nx.draw_networkx_nodes(G,pos,node_size=1000,nodelist=['A','C','D','E'],node_color='b')
nx.draw_networkx_edges(G,pos,edge_color='k',alpha=1,width=3)
nx.draw_networkx_labels(G,pos,labels,font_size=8,font_color='w')
nx.draw_networkx_edge_labels(G,pos,edge_labels={('A', 'B'):'Calle1',('B', 'D'):'Calle2',('B', 'E'):'Calle3',('A', 'C'):'Calle4'})
```

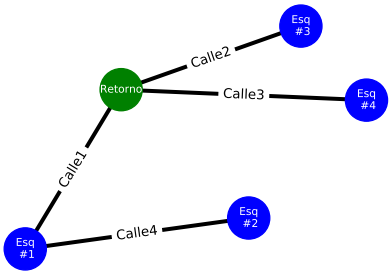

Figura 3: *Conexión entre avenidas y un retorno*

A pesar de que la representación es aceptable, no podemos decir que este algoritmo sea muy bueno, ya que así como un algoritmo random da una buena solución puede no darla, de hecho, si el código se corre varias veces, se obtienen diferentes versiones representativas, por lo que una vez hallado un dibujo que satisfaga al usuario, conserve la imagen con precaución.

4. Circular

En muchas ocasiones el tráfico vehicular se ve afectado por diversos factores haciendo que este sea lento, debido a ello en muchas calles solamente se permite el acceso en un solo sentido, así como se muestra en la figura 4.

Código 7: *Circular layout*

```
G = nx.DiGraph()
G.add_nodes_from(range(5))

pos = nx.circular_layout(G)

nx.draw_networkx_nodes(G,pos,node_size=600,nodelist=range(5),
    node_color='b')
nx.draw_networkx_edges(G,pos,edgelist=[(0,1),(1,2),(2,3),(2,4)
    ,(4,0)],edge_color='k',alpha=1,width=3,arrowsize=20)
nx.draw_networkx_labels(G,pos,{0:'A',1:'B',2:'C',3:'D',4:'E'},
    font_size=8,font_color='w')
nx.draw_networkx_edge_labels(G,pos,edge_labels={(0,1):'Calle1'
    ,(1,2):'Calle2',(2,3):'Calle3',(2,4):'Calle4',(4,0):'Calle5'
    })
```

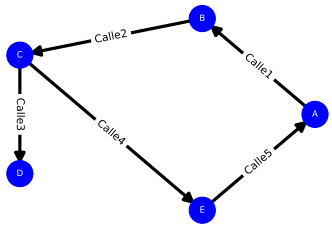

Figura 4: *Calles con sentido restringido*

La visualización que se obtuvo es realmente buena, ya que con la posición de los nodos y la introducción de etiquetas da una mejor representación de la realidad.

5. Pydot

Es importante modelar el como están hechas ciertas colonias en algunos países en el que suele haber una calle en la esquina de una localidad y es acompañada de un retorno o parque para facilitar la salida vehicular. En la figura 5 se muestra un ejemplo de esta situación.

Código 8: *Pydot layout*

```
G = nx.MultiDiGraph()
G.add_nodes_from(range(5))
G.add_edges_from([(0,1),(1,0),(1,2),(1,4),(2,3),(3,4),(4,0)
    ,(4,3)])

pos = nx.nx_pydot.pydot_layout(G)

nx.draw_networkx_nodes(G,pos,nodelist=[0,1,2,4],node_color='b',
    alpha=1,node_size=600)
nx.draw_networkx_nodes(G,pos,nodelist=[3],node_color='g',alpha
    =1,node_size=600)
nx.draw_networkx_edges(G,pos,edgelist=[(0,1),(1,0),(1,2),(1,4)
    ,(2,3),(3,4),(4,0),(4,3)],edge_color='k',alpha=1,width=3,
    arrowsize=30)
nx.draw_networkx_labels(G,pos,labels={0:'A',1:'B',2:'C',3:'D',4:
    'E'},font_size=12,font_color='k')
```

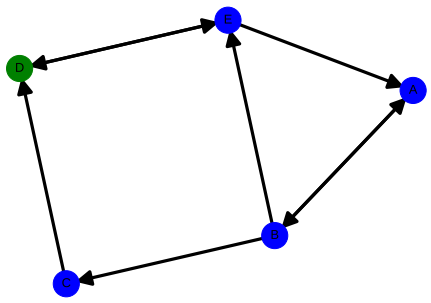

Figura 5: *Colonia*

.

6. Shell

Una fábrica que produce productos y los suministra a diferentes almacénes pero los vehículos pueden transitar no solamente por una carretera, sino por múltiples, en este caso 2. Ver figura 6.

Código 9: *Shell layout*

```
G = nx.MultiGraph()
G.add_nodes_from(range(5))

pos = nx.shell_layout(G)

nx.draw_networkx_nodes(G,pos,node_size=700,nodelist=[0],
    node_color='y',node_shape='s')
nx.draw_networkx_nodes(G,pos,node_size=700,nodelist=[1,2,3,4],
    node_color='b')
nx.draw_networkx_edges(G,pos,edgelist=[(0,1),(1,3)],edge_color='
    k',alpha=1,width=3)
nx.draw_networkx_edges(G,pos,edgelist=[(1,2),(3,4)],edge_color='
    r',alpha=1,width=3)
nx.draw_networkx_labels(G,pos,{0:'Dep',1:'A',2:'Suc \n#1',3:'B'
    ,4:'Suc \n#2'},font_size=8,font_color='w')
```

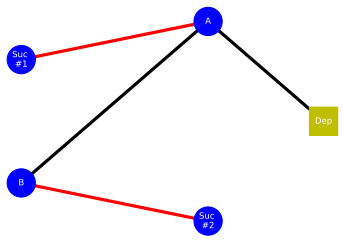

Figura 6: *Ruta de un almacén*

Este tipo de situación es de alta exigencia ya que en la práctica no se cuentan con pocos nodos, sino una gran cantidad y algo que puede considerarse es sustituir los nodos por las fotografías de las sucursales.

7. Fruchterman reingold

Un problema interesante en la práctica es donde se ven involucrados puentes. Una característica resaltante de este grafo es que se puede llegar de un vértice a cualquier otro por la conexión de las aristas.

Código 10: *Fruchterman reingold layout*

```
G=nx.MultiDiGraph()

G.add_nodes_from(range(5))
pos = nx.fruchterman_reingold_layout(G)

nx.draw_networkx_nodes(G,pos,node_size=500,nodelist=[0],
    node_color='g')
nx.draw_networkx_nodes(G,pos,node_size=500,nodelist=[1,2,3,4],
    node_color='b')
nx.draw_networkx_edges(G,pos,edgelist=[(1,2),(2,1),(2,3),(3,2)
    ,(4,0)],edge_color='k',alpha=1,width=3,arrowsize=20)
nx.draw_networkx_edges(G,pos,edgelist=[(3,4),(0,1),(1,0)],
    edge_color='r',alpha=1,width=3)

nx.draw_networkx_labels(G,pos,{0:'A',1:'B',2:'C',3:'D',4:'E'},
    font_size=8,font_color='w')
```

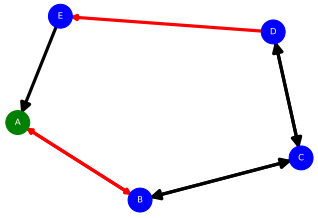

Figura 7: *Puentes de una ciudad*

Difícilmente los puentes tienen esta estructura, más bien son paralelos entre sí, pero esta representación puede ayudar en el caso en que se desea realizar un análisis para algún recorrido (como el problema de los 7 puentes).

8. Graphviz

Supóngase que en una colonia privada se tiene la libertad de transitar libremente por cualquier calle, donde al fondo hay un parque y este está en una calle que empieza y termina donde mismo, y además se puede obtener acceso a la avenida principal mediante dos calles. La figura 8 representa esta situación.

Código 11: *Graphviz layout*

```
G = nx.Graph()
G.add_nodes_from(range(5))
G.add_edges_from([(0,1),(0,2),(0,3),(0,4),(1,2),(1,4),(2,3)
    ,(3,4),])

pos = nx.nx_pydot.graphviz_layout(G)

nx.draw_networkx_nodes(G,pos,nodelist=range(4),node_color='b',
    alpha=1,node_size=600)
nx.draw_networkx_nodes(G,pos,nodelist=[4],node_color='g',alpha
    =1,node_size=600)
nx.draw_networkx_edges(G,pos,edgelist=[(0,1),(0,2),(0,3),(0,4)
    ,(1,4),(2,3),(3,4)],edge_color='k',alpha=1,width=3)
nx.draw_networkx_edges(G,pos,edgelist=[(1,2)],edge_color='r',
    alpha=1,width=3)
nx.draw_networkx_labels(G,pos,labels={0:'A',1:'B',2:'C',3:'D',4:
    'E'},font_size=12,font_color='k')
```

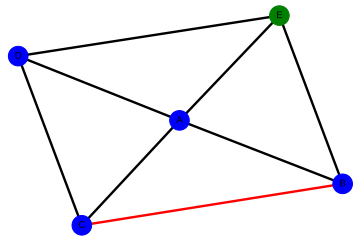

Figura 8: *Colonia con múltiples conexiones*

En la práctica cada colonia tiene su forma particular, pero esta representación es suficiente para lograr percibir la idea de la estructura de muchas colonias, así que para este ejemplo basta con esta representación.

9. Kamada kawai

Se desea llegar de una localidad a otra y se conocen los posibles caminos por los cuales se puede transitar y se desea encontrar la ruta que contenga menor costo para llevar a cabo el viaje.

Código 12: *Kamada kawai layout*

```
G = nx.MultiDiGraph()
G.add_nodes_from(range(5))

pos = nx.kamada_kawai_layout(G)

nx.draw_networkx_nodes(G,pos,node_size=400,nodelist=range(5),
    node_color='b')
nx.draw_networkx_edges(G,pos,edgelist=[(1,2),(1,3),(3,4)],
    edge_color='r',alpha=1,width=3)
nx.draw_networkx_edges(G,pos,edgelist=[(0,1),(0,2),(2,3),(2,4)],
    edge_color='k',alpha=1,width=3)
nx.draw_networkx_labels(G,pos,labels={0:'A',1:'B',2:'C',3:'D',4:
    'E'},font_size=12,font_color='w')
```

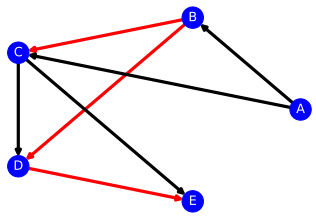

Figura 9: *Mapa transitorio*

La función de los grafos que representan redes de carreteras son más buenos por la información con la que pueden contar como etiquetas o pesos que por la visualizacion en sí, así que casi cualquier algoritmo de acomodo podría ser bueno para esta situación. Para este tipo de problema se puede proponer como trabajo a futuro el representar las carreteras sobre mapas.

10. Spring

Los problemas de ruteo de vehículos son de gran relevancia pues tienen muchas aplicaciones, dado que se puede modelar mediante un multigrafo dirigido ciclico por sus características físicas, se puede modelar como se muestra en la figura 10.

Código 13: *Spring layout*

```
G=nx.MultiDiGraph()
G.add_nodes_from(range(6))

pos = nx.spring_layout(G,iterations=300)

nx.draw_networkx_edges(G,pos,alpha=1,edgelist=[(0,2),(1,3),(1,2)
    ,(2,3),(2,4),(3,1),(3,4),(3,5),(4,2),(4,5)],width=1,arrowsize
    =20)
nx.draw_networkx_edges(G,pos,alpha=1,edgelist=[(0,1),(1,0)],
    width=1,edge_color='r',arrowsize=20)
nx.draw_networkx_nodes(G,pos,node_size=400,nodelist
    =[0,1,2,3,4,5],node_color='b')
nx.draw_networkx_labels(G,pos,{0:'A',1:'B',2:'C',3:'D',4:'E',5:'
    F'},font_size=8,font_color='w')
```

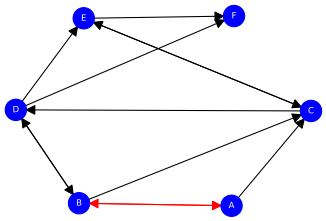

Figura 10: *Calles principales de una ciudad*

El algoritmo empleado para este grafo se aplica muy bien ya que trata de esparcir todos los nodos de forma equitativa dentro de un área, y dependiendo del número de nodos.

11. Force atlas 2

La conectividad de cables y circuitos en una computadora, tanto los cables entrantes como por ejemplo los de encender y de internet, los circuitos mismos en el cpu, y los cables de salida como teclado, mouse, etc. Dicha representación de la red computacional se encuentra en la figura 11.

Código 14: *Force atlas layout*

```
forceatlas2 = ForceAtlas2(
                          outboundAttractionDistribution=True,
                          linLogMode=False,
                          adjustSizes=False,
                          edgeWeightInfluence=1.0,

                          jitterTolerance=1.0,
                          barnesHutOptimize=True,
                          barnesHutTheta=1.2,
                          multiThreaded=False,

                          scalingRatio=2.0,
                          strongGravityMode=False,
                          gravity=1.0,

                          verbose=True)

positions = forceatlas2.forceatlas2_networkx_layout(G, pos=None,
    iterations=200)
```

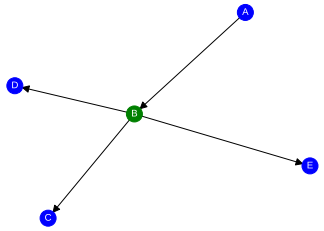

Figura 11: *Red de conexión computacional*

Este algoritmo proporciona una buena representación, el único detalle es que las aristas están muy largas, provocando que el grafo sea extenso si se empiezan a considerar mayor cantidad de nodos.

Referencias

[1] Python. https://www.python.org/.

[2] G. Mario. https://github.com/MarioGtz14/FlujoRedesMario.

[3] https://networkx.github.io/documentation/networkx1.10/reference/classes.html.

[4] Matplotlib. https://matplotlib.org/.

[5] PyGraphviz. https://pygraphviz.github.io/documentation/latest/tutorial.html.