Visualización de grafos

Brenda Yaneth Sotelo Benítez
5705

26 de febrero de 2019

Este trabajo busca visualizar grafos utilizando un diferente algoritmo de acomodo (inglés: layout) para cada caso, con la implementación de la librería Networkx [1] y Matplotlib [2] de `Python` [3] para la generación de grafos y guardar imágenes en el formato *eps* respectivamente. El código empleado se obtuvo consultando documentación oficial [4] y los ejemplos mostrados se encuentran en [5]. Se utilizan arcos rojos para representar cuando un nodo tiene múltiples aristas y un nodo cuadrado verde para representar un lazo.

1. Grafo simple no dirigido acíclico

El algoritmo de acomodo utilizado para este ejemplo es el `kamada_kawai_layout` que toma en cuenta las distancias cuando posiciona un nodo. Se observa que el resultado de aplicarlo es bueno debido a que la cantidad de cruces es mínima, se mantiene la idea del ejemplo original y no es necesario hacer iteraciones. El resultado de la visualización se muestra en la figura 1.

```
1  G=nx.Graph()
2
3  A=[(1,2),(2,3),(3,4),(4,5),(2,6),(2,7),(3,8),(3,9),(4,10),(4,11)]
4  G.add_edges_from(A)
5
6  pos = nx.kamada_kawai_layout(G)
7
8  nx.draw_networkx_nodes(G,pos,nodelist=[1,2,3,4,5,6,7,8,9,10,11],
       node_color='gray', node_size=500)
9  nx.draw_networkx_edges(G,pos,width=3,edgelist=A, )
10
11 plt.axis('off')
12 plt.savefig("GNA.eps")
```

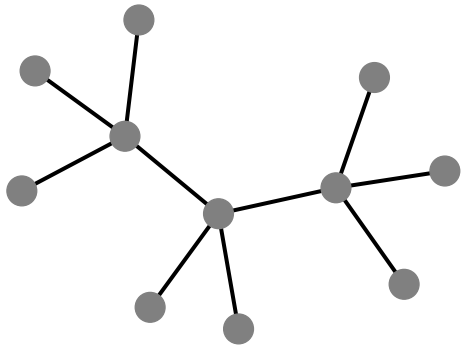

Figura 1: *Propano (C_3H_8)*

2. Grafo simple no dirigido cíclico

En la figura 2 se puede apreciar el resultado de aplicar el algoritmo de `fruchterman_reingold_layout` que al igual que `kamada_kawai_layout` son dirigidos por fuerzas, con la característica especial de la distribución de los nodos de forma homogénea. Se fijo un total de mil iteraciones para que el grafo se ajustara sin perder la forma debido a que con la ejecución de menos iteraciones el resultado es malo.

```
1  G=nx.Graph()
2
3  A=[(0,1),(1,2),(2,3),(3,4),(4,5),(0,5),(1,4),(0,6),(6,7)]
4
5  G.add_edges_from(A)
6  labels={2:'1',3:'2',4:'3',5:'4',0:'5',1:'6',6:'7',7:'8'}
7  pos= nx.fruchterman_reingold_layout(G, iterations=1000)
8
9  nx.draw_networkx_nodes(G,pos,nodelist=[0,1,2,3,4,5,6,7],node_color=
       'b', node_size=300)
10 nx.draw_networkx_edges(G,pos,width=1,edgelist=A,alpha=1)
11 nx.draw_networkx_labels(G,pos,labels, font_size=12, font_color='w'
       )
12
13 plt.axis('off')
```

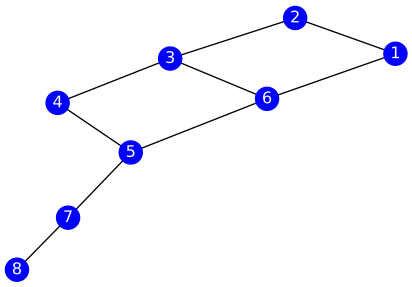

Figura 2: *Red de computadoras*

3. Grafo simple no dirigido reflexivo

El algoritmo `shell_layout` utilizado para la visualización del ejemplo de calles con retorno, tal y como se muestra en la figura 3 tiene buenos resultados a comparación del ejemplo original y de los otros diseños de acomodo utilizados, destacando que posiciona los nodos en círculos concéntricos y tiene la facilidad de elegir el centro del diseño. A continuación se muestra parte del código realizado donde también se hace uso de formas diferentes en los nodos y aristas.

```
1  G=nx.Graph()
2  A=[(0,1),(1,2),(2,3),(3,4)]
3  G.add_edges_from(A)
4  labels={1:'R'}
5  pos=nx.shell_layout(G)
6
7  nx.draw_networkx_nodes(G,pos,nodelist=[1],node_shape='s',
       node_color='g',node_size=500)
8  nx.draw_networkx_nodes(G,pos,nodelist=[3,4],node_color='b',
       node_size=80)
9  nx.draw_networkx_nodes(G,pos,nodelist=[0,2],node_color='b',
       node_size=80)
10 nx.draw_networkx_edges(G,pos,width=3,edgelist=[(2,3),(3,4)],alpha
       =1,style='dashed')
11 nx.draw_networkx_edges(G,pos,width=3,edgelist=[(0,1),(1,2)],alpha
       =1,style='dashed')
12 nx.draw_networkx_labels(G,pos,labels, front_size=12)
13 nx.draw_networkx_edge_labels(G, pos,font_family='sans-serif',
       edge_labels={(0,1):'Barragan',(1,2):'Gonzalitos',(2,3):'Ruiz␣
       Cortines',(3,4):'Rangel␣Frias'})
14
15 plt.axis('off')
16 plt.savefig("GNR.eps")
```

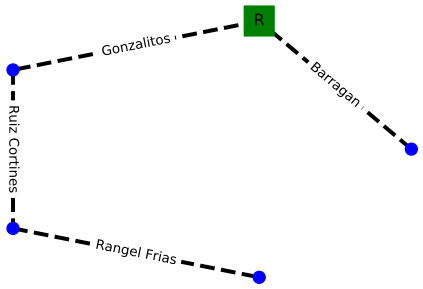

Figura 3: *Representación de calles con un retorno*

4. Grafo simple dirigido acíclico

El algoritmo de acomodo `circular_layout` como su nombre lo indica, tiene la característica de acomodar los nodos en círculo. Debido a esto, el ejemplo de secuencia de tareas que se muestra en la figura 4 tiene un buen ajuste en la representación e incluso mejor que el original. Este diseño es útil para crear grafos circulares con un número mayor de nodos.

```
1  G=nx.DiGraph()
2
3  A=[(0,1),(1,2),(2,3),(3,4),(3,5),(3,6),(4,7),(5,8),(8,6),(7,9)
       ,(9,10),(6,9),(1,3)]
4  labels={0:'1',1:'2',2:'3',3:'4',4:'5',5:'6',6:'9',7:'7',8:'8',9:'10
       ',10:'11'}
5  G.add_edges_from(A)
6
7  pos =  nx.circular_layout(G)
8
9  nx.draw_networkx_nodes(G,pos,nodelist=[0,1,2,3,4,5,6,7,8,9,10],
       node_color='m', node_size=450, alpha=0.8)
10 nx.draw_networkx_edges(G,pos,width=1.2,edgelist=A,alpha=1,arrowsize
       =20)
11 nx.draw_networkx_labels(G,pos,labels, front_size=12)
12
13 plt.axis('off')
14 plt.savefig("GDA.eps")
```

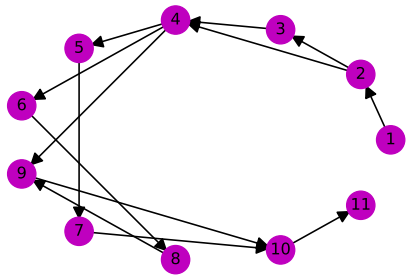

Figura 4: *Secuencia de tareas*

5. Grafo simple dirigido cíclico

Para el diseño del ejemplo de ruteo de vehículos se hizo uso del algoritmo `spring_layout` que puede recibir hasta once parámetros y utiliza el algoritmo de Fruchterman-Reingold. El resultado de cien iteraciones se muestra en la figura 5 dado que a menor número de iteraciones los nodos están muy dispersos y hay cruces de aristas y a mayor número de iteraciones los nodos están más cercanos, es por eso que se tomó una cantidad media de iteraciones.

```
1  G=nx.DiGraph()
2  A=[(0,1),(1,2),(2,3),(3,4),(4,0),(0,5),(5,6),(6,7),(7,8),(8,0)]
3  G.add_edges_from(A)
4  labels={0:'D'}
5  pos=nx.spring_layout(G, iterations=100)
6  nx.draw_networkx_nodes(G,pos,nodelist=[0],node_shape='s',node_color
       ='b', node_size=300)
7  nx.draw_networkx_nodes(G,pos,nodelist=[1,2,3,4],node_color='pink')
8  nx.draw_networkx_nodes(G,pos,nodelist=[5,6,7,8],node_color='c')
9  nx.draw_networkx_edges(G,pos,width=2,edgelist=A,alpha=1,arrowsize
       =25)
10 nx.draw_networkx_labels(G,pos,labels,front_size=12,font_color='w')
11
12 plt.axis('off')
13 plt.savefig("GDC.eps")
```

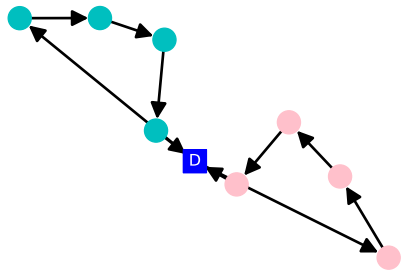

Figura 5: *VRP*

6. Grafo simple dirigido reflexivo

En la figura 6 se muestra el resultado de aplicar el algoritmo de acomodo `random_layout` que coloca los vértices al azar en un cuadrado al ejemplo de representación de una tienda proveedora que vende productos a varias sucursales pequeñas y que a su vez vende para si misma. Este algoritmo fue difícil de decidir donde aplicarlo ya que presenta una gran cantidad de cruces en las aristas por lo que se ejecutó el código varias veces para obtener la representacón que más se ajustara.

```
1  G=nx.DiGraph()
2  A=[(0,1),(0,2),(0,3),(0,4),(3,4)]
3  labels={0:'P',1:'T1',2:'T2',3:'T3',4:'T4'}
4  G.add_edges_from(A)
5  pos=nx.random_layout(G)
6
7  nx.draw_networkx_nodes(G,pos,nodelist=[1,2,3,4],node_color='c',
       node_size=500)
8  nx.draw_networkx_nodes(G,pos,nodelist=[0],node_color='g', node_size
       =1000,node_shape='s')
9  nx.draw_networkx_edges(G,pos,width=1,edgelist=A,alpha=1,arrowsize
       =20)
10 nx.draw_networkx_labels(G,pos,labels, front_size=12,font_color='w')
11
12 plt.axis('off')
13 plt.savefig("GDR.eps")
```

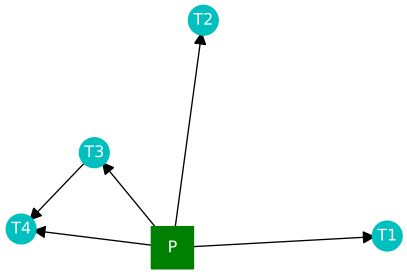

Figura 6: *Red de distribución*

7. Multigrafo no dirigido acíclico

El camino que sigue un río cuando pasa por ciertas localidades se puede representar mediante un multigrafo tal como se observa en figura 7. Esto debido a que un río suele dividirse cuando tiene de por medio obstáculos como tierra, piedras y se disparse en diferentes caminos beneficiando a varias comunidades. El algoritmo de acomodo que se ajusta a esta representación es `nx_pydot.pydot_layout` que se muestra en la figura 7.

```
1  G=nx.MultiGraph()
2  G.add_nodes_from([0,1,2,3,4,5,6])
3
4  pos = nx.nx_pydot.pydot_layout(G)
5
6  A=[(0,1),(1,2),(0,3),(2,4),(3,5),(2,6)]
7
8  nx.draw_networkx_nodes(G,pos,nodelist=[0,1,2,3,4,5,6],node_color='b
       ')
9  nx.draw_networkx_edges(G,pos,width=1,edgelist=A,alpha=1,edge_color=
       'c')
10 nx.draw_networkx_edges(G,pos,width=3,edgelist=[(0,3)],alpha=1,
       edge_color='r')
11 nx.draw_networkx_labels(G,pos,front_size=12)
12
13 plt.axis('off')
14 plt.savefig("MNA.eps")
15 plt.show()
```

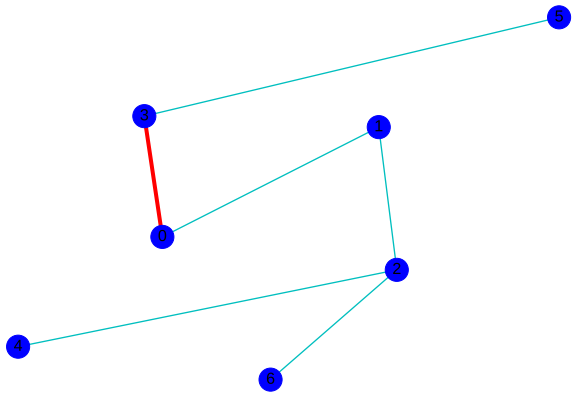

Figura 7: *Representación de un río*

8. Multigrafo no dirigido cíclico

Para la representación del ejemplo de los aeropuertos donde existen diversas aerolíneas y cada una de ellas cuenta con cierta cantidad de aviones que parten a su destino y regresan al aeropuerto, en la figura 8 se muestra el resultado de aplicar el algoritmo `bipartite_layout` que coloca los nodos en dos líneas rectas y se obtiene un buen resultado en comparación con el original. Es de utilidad para representar árboles.

```
G=nx.MultiGraph()
A=[(1,2),(2,3),(3,4),(3,1),(3,2),(2,4),(1,3)]
G.add_edges_from(A)
labels={1:'MÃ©xico', 3: 'Nayarit', 4: 'CancÃºn' }
labels1={2:'Monterrey'}
pos = nx.bipartite_layout(G,{1,4}, scale=0.2)

nx.draw_networkx_nodes(G,pos,nodelist=[1,2,3,4],node_color='b',
    node_shape='p')
nx.draw_networkx_edges(G,pos,width=1,edgelist=A,alpha=1)
nx.draw_networkx_edges(G,pos,width=3,edgelist=[(2,3)],alpha=1,
    edge_color='r',size=0.1)
pos1=pos
for i in pos:
    pos1[i][1]=pos1[i][1]+0.04
nx.draw_networkx_labels(G,pos,labels, font_size=12)
for i in pos:
    pos1[i][1]=pos1[i][1]-0.08
nx.draw_networkx_labels(G,pos,labels1, font_size=12)
```

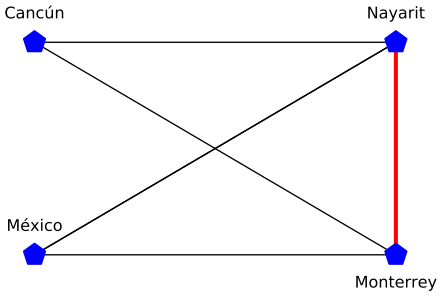

Figura 8: *Vuelos de aerolíneas*

9. Multigrafo no dirigido reflexivo

En la figura 9 se muestra el algoritmo `forceatlas2_layout` para el ejemplo de un inversionista que tiene a varios socios. Él puede decidir si invertir, que inviertan en él o invertir en si mismo, lo cual esto último crearía el lazo en el grafo.

```
G = nx.MultiDiGraph()
G.add_nodes_from(range(6))
labels={1:'I', 2:'S1',3:'S2',4:'S3',5:'S4'}
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)

pos = forceatlas2.forceatlas2_networkx_layout(G, pos=None,
    iterations=100)




nx.draw_networkx_nodes(G, pos, nodelist=[2,3,4,5], node_size=400,
    with_labels=True, node_color="m")
nx.draw_networkx_nodes(G, pos, nodelist=[1], node_size=400,
```

```
    with_labels=True, node_color="g")
nx.draw_networkx_edges(G, pos,edgelist=[(1,5)],edge_color="r")
nx.draw_networkx_edges(G, pos,edgelist=[(1,2),(1,3),(1,4)],
    edge_color="k")
nx.draw_networkx_labels(G,pos,labels, front_size=12, font_color='w'
    )

plt.axis('off')
plt.savefig('MNR.eps')
```

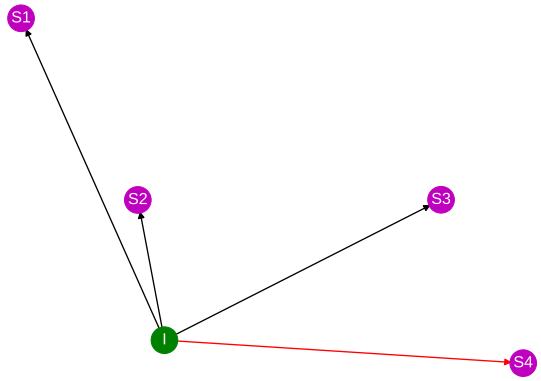

Figura 9: *Proceso de inversión*

10. Multigrafo dirigido acíclico

El algoritmo de acomodo utilizado para el ejemplo de los ductos de agua conectados de forma consecutiva sin formar ciclos es el `spectral_layout` que coloca los nodos utilizando vectores propios. De la misma forma que `random_layout` se presentó la dificultad de analizar en donde aplicarlo ya que los nodos y aristas se sobredibujaban y el grafo perdía forma.

```
1
2 A=[(1,2),(2,3),(3,4),(4,5)]
3 G.add_edges_from(A)
4
5 pos =nx.spectral_layout(G)
6
7 nx.draw_networkx_nodes(G,pos,nodelist=[1,2,3,4,5],node_color='b',
      node_size=300)
8 nx.draw_networkx_edges(G,pos,width=2,edgelist=A,alpha=1, edge_color
      ='c')
9 nx.draw_networkx_edges(G,pos,width=3,edgelist=[(1,2),(3,4)],alpha
      =1,edge_color='r')
10 nx.draw_networkx_edge_labels(G, pos,edge_labels={(1,2):'Ductos 1 y
      2',(2,3):'Ducto 3',(3,4):'Ductos 4 y 5',(4,5):'Ducto 6'})
11
12 plt.axis('off')
13 plt.savefig("MDA.eps")
```

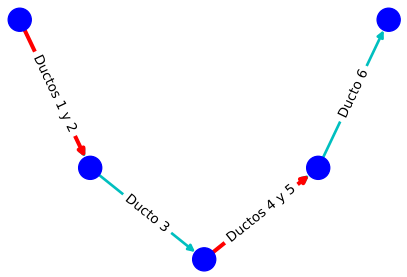

Figura 10: *Ductos de agua*

11. Multigrafo dirigido cíclico

En la figura 11 se aplica el algoritmo `shell_layout` en el problema representativo de calles o avenidas, donde para llegar a un lugar existen varias alternativas. El diseño muestra una buena visualización.

```
G=nx.MultiDiGraph()

A=[(1,2),(2,4),(4,3),(3,1),(5,2),(6,5),(7,5)]
G.add_edges_from(A)

pos =nx.shell_layout(G)

nx.draw_networkx_nodes(G,pos,nodelist=[1,2,3,4,5,6,7],node_color='y
    ', node_size=400,node_shape='^')
nx.draw_networkx_edges(G,pos,width=1,alpha=1,arrowsize=20)
nx.draw_networkx_edges(G,pos,width=3,edgelist=[(5,2)],alpha=1,
    edge_color='r', arrowsize=20)

plt.axis('off')
plt.savefig("MDC.eps")
```

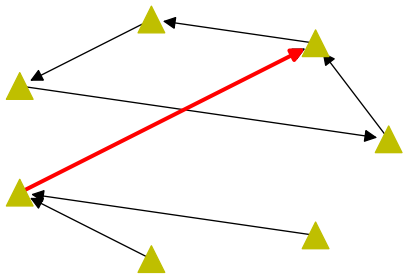

Figura 11: *Representación de avenidas*

12. Multigrafo dirigido reflexivo

En la figura 12 se plantea un modelo donde un hospital suministra medicinas a varias estaciones móviles, pero también se suministra el mismo. El algoritmo utilizado para la representación es el `nx_pydot.graphviz_layout`.

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

V = nx.nx_pydot.graphviz_layout(G)

labels={0:'Hospital'}

nx.draw_networkx_nodes(G,V, nodelist=[1,2,3,4], node_color='pink')
nx.draw_networkx_nodes(G,V, nodelist=[0], node_color='r')
nx.draw_networkx_edges(G,V, width=1,edgelist=[(0,1),(0,2),(0,3)
    ,(0,4)], alpha=1)
nx.draw_networkx_edges(G,V, width=3,edgelist=[(0,3)], alpha=1,
    edge_color='c')
nx.draw_networkx_labels(G,V, labels,  front_size=12)

plt.axis('off')
plt.savefig("MDR.eps")
```

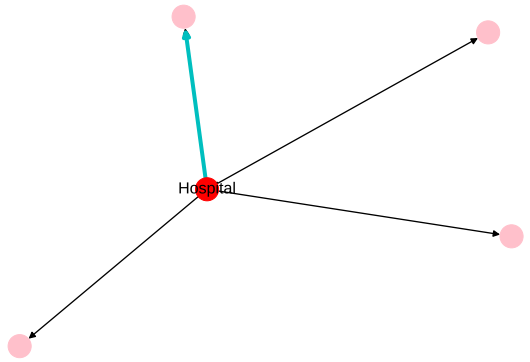

Figura 12: *Distribución de ayuda*

Referencias

[1] NetworkX developers Versión 2.0. `https://networkx.github.io/`.

[2] The Matplotlib development team Versión 3.0.2. https://matplotlib.org/.

[3] Python Software Foundation Versión 3.7.2. https://www.python.org/.

[4] NetworkX developers con última actualización el 19 de Septiembre 2018. https://networkx.github.io/documentation/networkx-1.10/reference/drawing.html.

[5] Sotelo B. Repositorio optimización flujo en redes. https://github.com/BrendaSotelo/Flujo_Redes_BSotelo.