Tarea No.2: Rectificada

Dayli Machado (5275)

3 de junio de 2019

1. Grafo simple no dirigido acíclico usando *bipartite layout*

Un grafo simple no dirigido acíclico, es aquel que no posee dirección en sus aristas, ni bucles, y no es reflexivo [5]. Este tipo de grafos suele representarse por una secuencia de nodos unidos por aristas, donde de cada nodo generalmente solo sale o llega una arista [1].

Este tipo de grafos puede emplearse para representar estructuras moleculares que sigan este comportamiento, un ejemplo sería la representación de alcanos de tipo lineal, donde los átomos se representan por los nodos y los enlaces entre estos por las aristas. El tipo de algoritmo de acomodamiento (*layout*) que se utilizó para lograr esta representación fue el *bipartite layout* el cual permite posicionar los nodos en dos líneas rectas y trazar ejes entre ellos [4], a partir de tener dos conjuntos de nodos diferentes se puede graficar la relación entre ellos. Se observa en la línea de código número 28 la forma de representar esta función empleada para la representación del grafo. Ver figura 1 en la página 2.

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Feb 24 17:34:52 2019
@author: Dayli
"""
import matplotlib.pyplot as plt
import networkx as nx

G = nx.Graph()

G.add_node(2)
G.add_node(4)

G.add_node(1)
G.add_node(3)
G.add_node(5)

G.add_edges_from([(1,2), (3,4)])
G.add_edges_from([(2,3), (4,5)])

labels = {}
labels[1] = r'H3C'
labels[2] = r'H2C'
labels[3] = r'CH2'
labels[4] = r'H2C'
labels[5] = r'CH3'

pos = nx.bipartite_layout(G, {1,3,5},align='horizontal', scale=0.05)

nx.draw_networkx_nodes(G,pos, node_size=800, node_color='b')
nx.draw_networkx_edges(G,pos, width=3)
nx.draw_networkx_labels(G,pos, labels, font_size=10)
plt.axis('off')
plt.savefig("imagenes1/Fig01.eps")

plt.show()
```

grafo1laybi.py

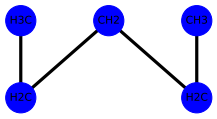

Figura 1: Molécula de alcano lineal usando *bipartite layout*

2. Grafo simple no dirigido cíclico usando *circular layout*

Un grafo simple no dirigido cíclico es aquel que no posee dirección en sus aristas, pero si se forma un ciclo que represente una figura cerrada que comience y termine en el mismo vértice, entonces es llamado cíclico [2]. En los grafos no dirigidos el flujo puede fluir en ambas direcciones.

Este tipo de grafos suele representarse por una secuencia de nodos unidos por aristas, pero estos pueden tener dentro un ciclo, o ser un ciclo en sí mismo. En la práctica pudiera ser la representación de las autopistas de una región dada, o también la representación de otro tipo de moléculas de alcanos que posean áreas cerradas en su representación, o las llamdas que se realizan interdepartamentales en una empresa.

Se toma de ejemplo las llamadas que se realizan entre los departamentos de producción, de marketing, comercial, calidad y planificación de la producción en una empresa, donde cada nodo representa un departamento y las aristas las llamadas que se realizan entre ellos. Para este tipo de grafo se emplea el *layout circular* como algoritmo de acomodamiento, pues permite ubicar los nodos en círculos [4]. De esta forma se visualiza mejor la distribución espacial de los nodos y la relación entre ellos que son las llamadas, pero como en este ejemplo la posición de los nodos puede ser aleatoria, se puede emplear también el *ramdom layout* como otro algoritmo de acomodamiento pues este permite ubicar los nodos uniformemente al azar en el cuadro [4]. En la figura 2 en la página 4 se muestra la representación gráfica del mismo.

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Feb 24 19:25:28 2019

@author: Dayli
"""

import matplotlib.pyplot as plt
import networkx as nx

G=nx.Graph()

G.add_nodes_from([0,1,2,3,4])

G.add_edge(0,1)
G.add_edge(0,3)
G.add_edge(1,4)
G.add_edge(1,2)
G.add_edge(3,1)
G.add_edge(0,4)
G.add_edge(2,4)

labels = {}
labels[0] = r'Markt'
labels[1] = r'Prod'
labels[2] = r'Cald'
labels[3] = r'Comerc'
labels[4] = r'Planif'

pos = nx.circular_layout(G, scale=0.5)
#pos = nx.spring_layout(G, scale=0.5)

nx.draw_networkx_nodes(G, pos, node_size=3000, node_color='#756b6b', node_shape='o')
nx.draw_networkx_edges(G, pos, width=2, alpha=0.5, edge_color='black')
nx.draw_networkx_labels(G, pos, labels, font_size=10, font_color='white')
plt.axis('off')
plt.savefig("imagenes1/Fig02.eps")
plt.show()
```

grafo2laycir.py

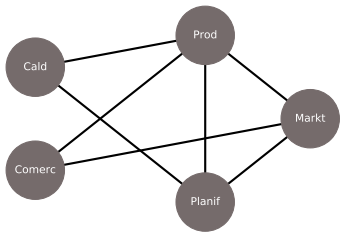

Figura 2: Llamadas interdepartamentales usando acomodo circular

3. Grafo simple no dirigido reflexivo usando *circular layout*

Un grafo simple no dirigido reflexivo es aquel en el cual no se permiten aristas múltiples, no posee dirección en sus aristas y cuenta al menos con un bucle, lo que se refiere a una arista reflexiva en la que coinciden el vértice de origen y el de destino [5].

Este grafo puede emplearse para representar el entrecruzamiento de grupos raciales, donde cada grupo representa un nodo diferente y las relaciones entre los grupos seran los hijos que tengan, siendo una relación no dirigida. Al relacionarse entre la misma raza estarían ocurriendo lazos reflexivos. Si se consideran los cuatro grupos representativos de las razas humanas según la clasificación de especialistas serían: blanco/caucásico, asiático/mongoloide, negroide/negro y australoide; esta clasificación posee dentro de cada grupo una clasificación de hasta 30 subgrupos, pero solo se representarán los cuatro principales. El *layout* que mejor refleja este tipo de grafo es el circular, la justificación es la misma explicada en la sección anterior. El código para representar este grafo fue tomado y adaptado del sitio [3], el mismo permitió insertar las imágenes como nodos. Ver figura 3 en la página 7.

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Feb 24 23:03:42 2019

@author: Dayli
"""
import networkx as nx
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# image from http://matplotlib.sourceforge.net/users/image_tutorial.html
path = "C:/Users/lapi7/Desktop/Para tarea 2/razas/"
img1=mpimg.imread("1.jpg")
img2=mpimg.imread("2.jpg")
img3=mpimg.imread("3.jpg")
img4=mpimg.imread("4.jpg")

G=nx.Graph()

G.add_nodes_from([1,2,3,4])

G.add_edges_from([(1,1), (1,2), (1,3), (1,4)])# Cada nodo es reflexivo
G.add_edges_from([(2,2), (2,1), (2,3), (2,4)])#
G.add_edges_from([(3,3), (3,1), (3,2), (3,4)])#
G.add_edges_from([(4,4), (4,1), (4,3), (4,2)])#

G.node[1]["image"]=img1
G.node[2]["image"]=img2
G.node[3]["image"]=img3
G.node[4]["image"]=img4

pos=nx.circular_layout(G, scale=0.5)

fig=plt.figure(figsize=(13,13))
ax=plt.subplot(1,1,1)
ax.set_aspect("equal")
nx.draw_networkx_edges(G,pos,ax=ax,width=2)

plt.xlim(-0.5,1.5)
plt.ylim(-0.5,1.5)

plt.axis("off")

trans=ax.transData.transform
trans2=fig.transFigure.inverted().transform

piesize=0.2 # this is the image size
p2=piesize/2.0
for n in G:
    xx,yy=trans(pos[n]) # figure coordinates
    xa,ya=trans2((xx,yy)) # axes coordinates
    a = plt.axes([xa-p2,ya-p2, piesize, piesize])
    a.set_aspect("equal")
    a.imshow(G.node[n]["image"])
    a.axis("off")

plt.savefig("imagenes1/Fig03.eps", bbox_inches='tight')
```

```
58 #nx.draw(G)
59 plt.show()
```

grafo3laycir.py

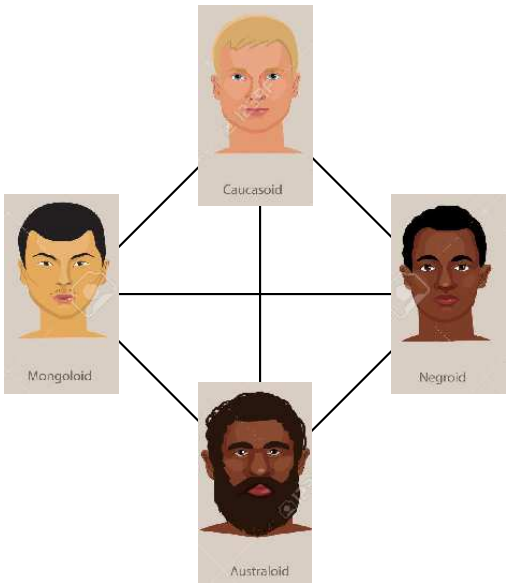

Figura 3: Grupos raciales usando *circular layout*

4. Gráfo simple dirigido acíclico usando *spring layout*

Un grafo simple dirigido acíclico, es aquel que posee una dirección en sus aristas, y no posee bucles o reflexividad, ni ciclos [5].

En este caso, ejemplos de la vida real son aquellos que poseen un origen del que puede salir una o varias aristas por diferentes caminos sin que entre ellas existan ciclos y posean un destino final diferente al del origen.

Los árboles genealógicos, los organigramas en las empresas, el flujo de procesos industriales que no posean ciclos, con un inicio y un fin, pudieran resultar ser ejemplos de la vida real en los que se puede aplicar este tipo de grafos.

Como algoritmo de acomodo para representar este grafo se usó el *spring layout*, el cual permite representar de manera sencilla las aristas rectas que salen de un nodo a otro y fijar la posición haciendo uso de sus parámetros [4] siendo el mismo algoritmo empleado en la tarea anterior para este tipo de grafo, sin embargo en este ejercicio se profundizó en el uso de sus parámetros.

Para la representación inicial se fija la posición de los nodos y el centro del grafo, además se estableció una distancia estándar entre ellos, también se emplea el degradado de color para indicar la transformación que recibe un objeto dado en una línea de producción. Ver figura 4 en la página 10.

```
import matplotlib.pyplot as plt
import networkx as nx

G=nx.DiGraph ()

G.add_node(1)
G.add_node(2)
G.add_node(3)
G.add_node(4)
G.add_node(5)

nod1={1}
nod2={2}
nod3={3}
nod4={4}
nod5={5}

G.add_edge(1,2)
G.add_edge(2,3)
G.add_edge(3,4)
G.add_edge(4,5)

positions = { 5:(0, 10), 4:(0, 20), 3:(0,30), 2:(0,40), 1:(0,50)}
#fixeds={1,2}
pos = nx.spring_layout(G, k=3, fixed=None, pos=positions, center=(0,0), iterations
    =50)

nx.draw_networkx_nodes(G,pos,nodelist=nod1,node_size=400,node_color='#580cf0',
    node_shape='o')
nx.draw_networkx_nodes(G,pos,nodelist=nod2,node_size=400,node_color='#6332c5',
    node_shape='o')
nx.draw_networkx_nodes(G,pos,nodelist=nod3,node_size=400,node_color='#6240a7',
    node_shape='o')
nx.draw_networkx_nodes(G,pos,nodelist=nod4,node_size=400,node_color='#574184',
    node_shape='o')
nx.draw_networkx_nodes(G,pos,nodelist=nod5,node_size=400,node_color='#433958',
    node_shape='o')


nx.draw_networkx_edges(G, pos, width=2, alpha=0.8, edge_color='black')
plt.axis('off')
#plt.axis('off')
plt.savefig("imagenes1/Fig04.eps")
plt.show()

#plt.show()
```

grafo4lyspring.py

5. Gráfo simple dirigido cíclico usando *bipartite layout*

Un grafo simple dirigido cíclico es aquel que posee una dirección en sus aristas y una figura cerrada que comienza y termina en el mismo vértice.

Figura 4: Transformación de un objeto en una línea de producción usando *spring layout*

Para este caso ejemplos de la vida real son aquellos que poseen un origen del que puede salir una o varias aristas por diferentes caminos, y que en un determinado nodo regresan a alguno anterior de modo que se forme uno o más ciclos. El destino final suele ser diferente al del origen.

Como aplicación real de este tipo de grafo se explica el flujo de procesos industriales o artesanales que posean ciclos, por ejemplo: la elaboración artesanal de jugo de naranja, al representar este proceso en un flujograma mediante un diagrama OPERIN, en el que se reflejen sus operaciones. Una vez que se llegue al paso de exprimir las naranjas, se cae en un ciclo volviendo a la operación anterior de seleccionar otra y otra naranja hasta cubrir toda la capacidad del extractor de jugo. En esta segunda tarea el algoritmo de acomodo empleado que se usó para representarlo fue el *bipartite layout* pues resulta más sencillo y directo para graficar los nodos que salen de la línea de producción y que forman el ciclo con los que se mantienen en ella, se emplea el parámetro *top* para separar el conjunto de nodos que no están en la línea principal de producción. Con este algoritmo se aprecia una mejor representación del grafo, aunque es válido aclarar que este ejemplo no se comporta exactamente como un grafo bipartito, pero este fue el algoritmo de acomodo que mejor lo representó. En este ejemplo las operaciones son los nodos y los enlaces la transformación que va teniendo la naranja como se muestra en la figura 5 en la página 12.

```python
import matplotlib.pyplot as plt
import networkx as nx

G=nx.DiGraph ()

G.add_node(1)
G.add_node(2)
G.add_node(3)

G.add_node(6)
G.add_node(4)
G.add_node(5)

G.add_edge(1,2)
G.add_edge(2,3)
G.add_edge(3,4)

G.add_edge(4,5)
G.add_edge(5,6)
G.add_edge(3,6)

G.add_cycle([5,3])

labels = {}
labels[1] = r'$Op1$'
labels[2] = r'$Op2$'
labels[3] = r'$Op3$'
labels[4] = r'$Op4$'
labels[5] = r'$Op5$'
labels[6] = r'$Op6$'

pos = nx.bipartite_layout(G, {4,5},align='horizontal', scale=0.7)

nx.draw_networkx_nodes(G, pos, node_size=500, node_color='r', node_shape='o')
nx.draw_networkx_edges(G, pos, width=2, alpha=0.8, edge_color='black')
nx.draw_networkx_labels(G, pos, labels, font_size=10)

plt.axis('off')
plt.savefig("imagenes1/Fig05.eps")
plt.show()
```

grafo5laybi.py

6. Gráfo simple dirigido reflexivo usando *circular layout*

La diferencia con este grafo y el anterior reflexivo visto, es que en este caso las aristas sí tienen sentido y el flujo que se analice debe ir en una dirección. En este caso deberá existir un nodo que se llame a sí mismo, o sea, que posea al menos un bucle.

Un ejemplo en el que se puede emplear la teoría de grafos de este tipo es para representar la disposición de personas con grupos sanguíneos diferentes a recibir sangre compatible o no. La reflexividad está dada en los casos en que cada grupo sanguíneo puede recibir de otros grupos y

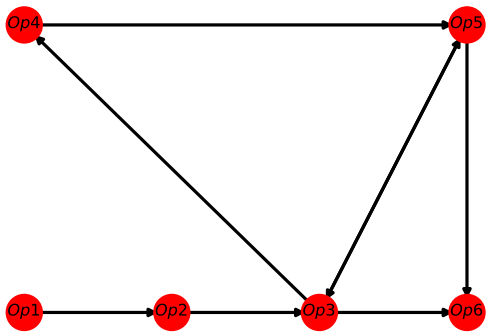

Figura 5: Flujo cíclico de una línea de producción artesanal de jugo de naranja

de sí mismo, excepto el grupo O- que sólo puede recibir del mismo tipo. Se cambia el algoritmo de acomodo anterior por el *circular layout* haciendo uso de sus parámetros, y se evidencia más estabilidad en las iteraciones de la figura que mantienen la misma posición deseada cada vez que se itera, sin tener que entrarla manualmente. Este se muestra en la figura 6 en la página 14.

```
import matplotlib.pyplot as plt
import networkx as nx

G = nx.DiGraph()
G.add_node(1)
G.add_node(2)
G.add_node(3)
G.add_node(4)
G.add_node(5)
G.add_node(6)
G.add_node(7)
G.add_node(8)

node_a = {1,2}
node_b = {3,4}
node_ab = {5,6}
node_o = {7,8}
#Se cambiaron los layout y el mejor resultado fue con el circular
#pos = nx.spring_layout(G)
pos = nx.circular_layout(G,dim=2,scale=1, center=(0,0))
#pos = nx.shell_layout(G)
#pos = nx.rescale_layout(G)

G.add_edges_from([(1,1), (8,1), (7,1), (2,1)])# Cada nodo es reflexivo pero no me sale la flechita
G.add_edges_from([(2,2), (8,2)])#igualmente no sale el reflexivo en si mismo
G.add_edges_from([(3,3), (8,3), (7,3), (4,3)])#no sale el reflexivo en si mismo
G.add_edges_from([(4,4), (8,4)])#es reflexivo en si mismo
G.add_edges_from([(5,5), (1,5), (2,5), (3,5), (4,5), (6,5), (7,5), (8,5)])# es reflexivo en si mismo
G.add_edges_from([(6,6), (4,6), (8,6), (2,6)])
G.add_edges_from([(7,7), (8,7)])
G.add_edges_from([(8,8)])

nx.draw_networkx_nodes(G, pos, nodelist=node_a, node_size=800, node_color='g', node_shape='o', alpha=0.3)
nx.draw_networkx_nodes(G, pos, nodelist=node_b, node_size=800, node_color='y', node_shape='o', alpha=0.3)
nx.draw_networkx_nodes(G, pos, nodelist=node_ab, node_size=800, node_color='b', node_shape='o', alpha=0.3)
nx.draw_networkx_nodes(G, pos, nodelist=node_o, node_size=800, node_color='r', node_shape='o', alpha=0.8)
#nx.draw_networkx_nodes(G, pos, nodelist=node_o, node_size=800, node_color='r', node_shape='on', alpha=0.1)
labels = {}
labels[1] = r'A+'
labels[2] = r'A-'
labels[3] = r'B+'
labels[4] = r'B-'
labels[5] = r'AB+'
labels[6] = r'AB-'
labels[7] = r'O+'
labels[8] = r'O-'
nx.draw_networkx_labels(G, pos, labels, font_size=10)

#pos1 = pos
#
```

```
#for key, value in G.edges:
#    print("(", pos[key][0] - 0.5, ", ", pos[key][1] - 0.5, ")\n")
#    pos1[key][0] = pos1[key][0] + 0.1
#    pos1[key][1] = pos1[key][1]
#

nx.draw_networkx_edges(G, pos, width=2, alpha=1, edge_color='black', scale=0.5 )
#print(pos)
plt.axis('on')
plt.savefig("imagenes1/Fig06.eps")
plt.show()
```

grafo6laycir.py

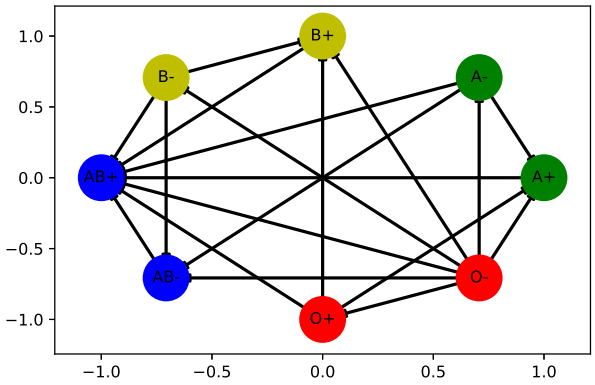

Figura 6: Compatibilidad entre grupos sanguíneos

7. Multigrafo no dirigido acíclico usando *spring layout*

Un multigrafo no dirigido acíclico es cuando existe más de una arista entre un par de vértices. Si no se permiten aristas múltiples, el grafo es simple [5]. Será no dirigido cuando las aristas no posean sentido restringido, sino que el flujo fluye en ambas direcciones, y en este caso no presenta ciclos.

Un ejemplo práctico de multígrafo no dirigido acíclico es la representación de determinados enlaces moleculares, como por ejemplo: un enlace de doble éster, como es el caso del éster sulfúrico. Para la representación de los elementos con este tipo de grafo se puede emplear el algoritmo de acomodo *spring layout*, es el mismo que se usó en el caso anterior pero esta vez se hace uso de sus parámetros sin fijar la posición [4], de este modo se obtiene una representación más ajustada a la realidad, también pudiera emplearse el *Kamada Kawai layout* pero al compararlos e iterar el *spring layaut* devuelve variantes más atractivas. El grafo usando *spring layout* se muestra en la figura 7 en la página 16.

```
import matplotlib.pyplot as plt
import networkx as nx

G=nx.MultiGraph()

G.add_node(1)
G.add_node(2)
G.add_node(3)
G.add_node(4)
G.add_node(5)
G.add_node(6)
G.add_node(7)

node_o = {4,5,6,7}
node_r = {1,2,3}

G.add_edge(4,3, weight=3)
G.add_edge(4,3, weight=5)
G.add_edge(3,7, weight=3)
G.add_edge(3,7, weight=5)
G.add_edge(1,5)
G.add_edge(5,3)
G.add_edge(3,6)
G.add_edge(6,2)

blue=[(4,3),(3,7),(1,5),(5,3),(3,6),(6,2)]
red=[(4,3),(3,7)]

labels = {}
labels[1] = r'R1'
labels[2] = r'R2'
labels[3] = r'S'
labels[4] = r'O'
labels[5] = r'O'
labels[6] = r'O'
labels[7] = r'O'

pos=nx.spring_layout(G, k=0.1, iterations=300, threshold=0.0001, weight='weight',
    scale=1)
nx.draw_networkx_nodes(G, pos, nodelist=node_o, node_size=600, node_color='b',
    node_shape='o')
nx.draw_networkx_nodes(G, pos, nodelist=node_r, node_size=600, node_color='y',
    node_shape='s')

nx.draw_networkx_edges(G, pos, edgelist=blue,width=6, alpha=0.5, edge_color='b',
    style='dashed')
nx.draw_networkx_edges(G, pos, edgelist=red,width=6, alpha=0.5, edge_color='r')

nx.draw_networkx_labels(G, pos, labels, font_size=10)

plt.axis('on')
plt.savefig("imagenes1/Fig07.eps")
plt.show()
```

grafo7lyspring.py

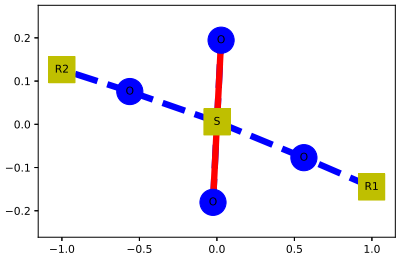

Figura 7: Mólecula de éster sulfúrio

8. Multigrafo no dirigido cíclico usando *Kamada Kawai layout*

Cuando exista más de una arista entre un par de vértices, el grafo se llama multígrafo. En este caso será no dirigido cuando las aristas no poseen sentido restringido, sino que el flujo fluye en ambas direcciones, y deberán existir zonas cerradas, o sea, que regresen al vértice inicial.

Este tipo de grafo es de gran utilidad para representar el comportamiento de redes sociales y neuronales [2]. El ejemplo práctico mostrado es cuando tenemos necesidad de comunicarnos con alguien por las vías de comunicación que tenemos en la actualidad, donde cada vía de comunicación seleccionada tendrá un costo (peso), y existen muchas vías para comunicarnos con otras personas, y a su vez para otras personas se comuniquen con nosotros, por lo que no es dirigido, pero sí cíclico, debido a los lazos que se pueden formar entre las personas que necesitan comunicarse. El algoritmo de acomodo empleado para este ejemplo según sus prestaciones fue el *Kamada Kawai layout* pues este tiene en cuenta para la representación el peso de la longitud del camino [4] y devuelve grafos más atractivos para el ejemplo en cuestión. El ejemplo se restringe a solo dos vías de comunicación, ya sea por *whatsap*, o por llamadas telefónicas, aunque en la práctica existen muchas más. Este grafo se muestra en la figura 8 en la página 18.

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 25 20:09:39 2019

@author: lapi7
"""

import matplotlib.pyplot as plot
import networkx as nx

G = nx.MultiGraph()

G.add_node(1)
G.add_node(2)
G.add_node(3)
G.add_node(4)
G.add_node(5)

G.add_edge(1,2, weight=3)
G.add_edge(1,2, weight=4)
G.add_edge(2,3, weight=3)
G.add_edge(2,3, weight=4)
G.add_edge(3,4, weight=3)
G.add_edge(3,4, weight=4)
G.add_edge(4,5, weight=3)
G.add_edge(4,5, weight=4)
G.add_edge(5,1, weight=3)
G.add_edge(5,1, weight=4)
G.add_edge(1,3, weight=3)
G.add_edge(1,3, weight=4)
G.add_edge(5,3, weight=3)
G.add_edge(5,3, weight=4)

green=[(1,2),(2,3),(3,4),(4,5),(5,1),(1,3),(5,3)]
yellow=[(1,2),(2,3),(3,4),(4,5),(5,1),(1,3),(5,3)]

pos = nx.kamada_kawai_layout(G, dist=None, pos=None, weight='weight', scale=0.5,
    center=None, dim=2)

plt.axis('off')

nx.draw_networkx_nodes(G, pos ,node_size=400, node_color='r', node_shape='o')

nx.draw_networkx_edges(G, pos, edgelist=green, width=3, alpha=0.5,
edge_color='g', style='dashed')
nx.draw_networkx_edges(G, pos, edgelist=yellow,width=4, alpha=0.5,
edge_color='y')

plt.savefig("imagenes1/Fig08.eps")

plt.show()
```

grafo8lykk.py

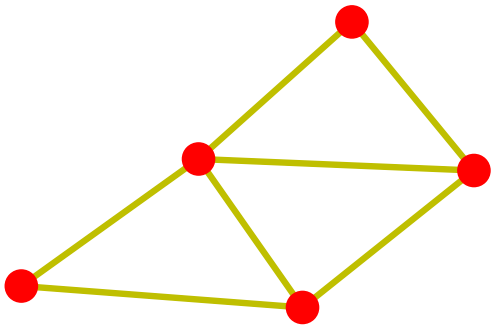

Figura 8: Vías de comunicación entre las personas

9. Multigrafo no dirigido reflexivo usando *random layout*

Este caso se refiere a un multígrafo en el que existe más de una arista entre un par de vértices, sin dirección entre estas, pero de un nodo deben salir más de una arista que regresen al mismo nodo, sin pasar por otro.

Una aplicación pudiera ser la relación que existe entre cinco signos zodiacales, donde cada nodo representa a las personas de un signo determinado y de cada uno de ellos puede salir una arista que represente las relaciones positivas entre los diferentes signos y otra las relaciones negativas, y a su vez las personas del mismo signo se relacionarán con sus iguales de manera positiva o negativa, lo que representa la reflexibilidad. Este es un grafo que no es exigente para su representación y aunque un algoritmo de acomodo circular puede ser apropiado, se propone usar uno aleatorio que funciona igual en este ejemplo y consume menos. Este grafo se muestra para cinco signos cualesquiera del zodiaco en la figura 9 en la página 20.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 11 16:09:42 2019

@author: lapi7
"""

import matplotlib.pyplot as plot
import networkx as nx

G=nx.MultiGraph()

G.add_node(1)
G.add_node(2)
G.add_node(3)
G.add_node(4)
G.add_node(5)

G.add_edge(1,2, weight=4)
G.add_edge(2,3, weight=3)
G.add_edge(2,3, weight=4)
G.add_edge(3,4, weight=3)
G.add_edge(3,4, weight=4)
G.add_edge(4,5, weight=3)
G.add_edge(4,5, weight=4)
G.add_edge(5,1, weight=3)
G.add_edge(5,1, weight=4)
G.add_edge(1,3, weight=3)
G.add_edge(1,3, weight=4)
G.add_edge(5,3, weight=3)
G.add_edge(5,3, weight=4)

node_A = {1}
node_S = {2}
node_E = {3}
node_V = {4}
node_C = {5}


green=[(1,2),(2,3),(3,4),(4,5),(5,1),(1,3),(5,3)]
yellow=[(1,2),(2,3),(3,4),(4,5),(5,1),(1,3),(5,3)]

pos = nx.random_layout(G, dim=2, center=None)  # Se puede usar el random en este
    caso
#pos = nx.circular_layout(G, dim=2, center=None)

plt.axis('off')

nx.draw_networkx_nodes(G, pos, nodelist=node_A, node_size=300, node_color='g',
    node_shape='o')
nx.draw_networkx_nodes(G, pos, nodelist=node_S, node_size=300, node_color='y',
    node_shape='o')
nx.draw_networkx_nodes(G, pos, nodelist=node_E, node_size=300, node_color='b',
    node_shape='o')
nx.draw_networkx_nodes(G, pos, nodelist=node_V, node_size=300, node_color='r',
    node_shape='o')
nx.draw_networkx_nodes(G, pos, nodelist=node_C, node_size=300, node_color='black',
```

```
    node_shape='o')
53
54
55 nx.draw_networkx_edges(G, pos, edgelist=green, width=3, alpha=0.5,
56 edge_color='g', style='dashed')
57 nx.draw_networkx_edges(G, pos, edgelist=yellow,width=4, alpha=0.5,
58 edge_color='y')
59
60 plt.savefig("imagenes1/Fig09.eps")
61
62 plt.show()
```

grafo9lyrandom.py

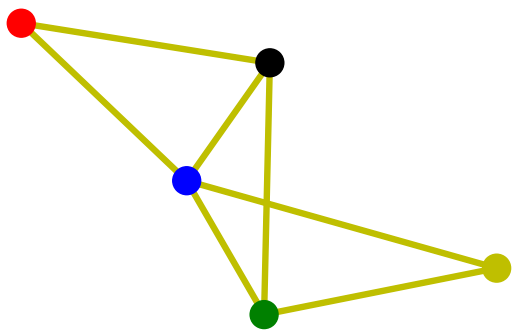

Figura 9: Relación entre personas del mismo signo zodiacal

10. Multigrafo dirigido acíclico usando *spectral layout*

Para definir un multigrafo dirigido acíclico debe existir más de una arista entre un par de nodos, con dirección y no formar ninguna figura cerrada dentro del grafo.

Este ejemplo aplicado a la práctica puede ser la representación de un viajero que desea ir de una ciudad a otra para visitarlas y tiene que elegir entre varias rutas posibles para llegar, o pudiera elegir también entre diferentes medios de transporte para trasladarse entre las ciudades, o una combinación de ambos. En este caso las ciudades serían los nodos, y las rutas o medios de transporte posibles a elegir entre un nodo u otro serían las aristas. Una ruta pudiera ser ir de la ciudad de Matanzas en Cuba, a la Habana, de esta a Monterrey, de Monterrey a Torreón y de ahí a Sinaloa. Para devolver el acomodo de estos nodos puede emplearse el algoritmo *spectral layout* que no se ha empleado en las secciones anteriores, este devuelve una forma sinusoidal pues refleja el algoritmo basado en la utilización de los vectores del gráfico laplaciano [4], lo cual no afecta a la visualización del grafo ejemplo. Asimismo se puede emplear el *Kamada Kawai layout* con resultados gráficos

más direccionados y apropiados para el ejemplo, pero ya se usó en secciones anteriores. El grafo se muestra en la figura 10 en la página 22.

```
import matplotlib.pyplot as plt
import networkx as nx

G=nx.MultiDiGraph()

G.add_node(1)
G.add_node(2)
G.add_node(3)
G.add_node(4)
G.add_node(5)

G.add_edge(1,2, weight=3)
G.add_edge(1,2, weight=4)
G.add_edge(2,3, weight=3)
G.add_edge(2,3, weight=4)
G.add_edge(3,4, weight=3)
G.add_edge(3,4, weight=4)
G.add_edge(4,5, weight=3)
G.add_edge(4,5, weight=4)

medio=[(1,2),(2,3),(3,4),(4,5)]
ruta=[(1,2),(2,3),(3,4),(4,5)]

#pos = nx.spring_layout(G)
#pos=nx.kamada_kawai_layout(G, dist=None, pos=None, weight='weight', scale=0.5,
    center=None, dim=2)

pos=nx.spectral_layout(G, weight='weight', scale=1, center=None, dim=2)

nx.draw_networkx_nodes(G, pos, node_size=800, node_color='#ecf815', node_shape='o')

nx.draw_networkx_edges(G, pos, edgelist=medio, width=6, alpha=0.8,
edge_color='black', style='dashed')

nx.draw_networkx_edges(G, pos, edgelist=ruta,width=4, alpha=0.5,
edge_color='r')

labels = {}
labels[1] = r'Mat'
labels[2] = r'Hab'
labels[3] = r'Monty'
labels[4] = r'Torr'
labels[5] = r'Sing'

plt.axis('off')

nx.draw_networkx_labels(G, pos, labels, font_size=10)

plt.savefig("imagenes1/Fig10.eps")
plt.show()
```

grafo10lyspectral.py

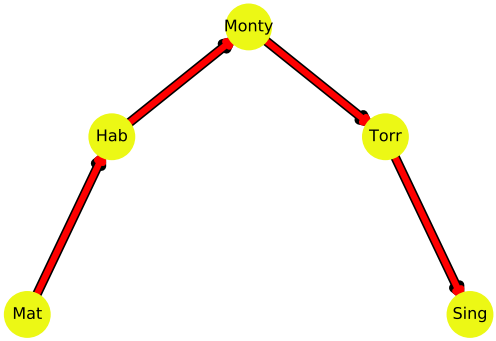

Figura 10: Ruta para viajar de Matanzas a Sinaloa

11. Multigrafo dirigido cíclico usando *spring layout*

En este tipo de grafos dirigidos cíclicos debe existir más de una arista entre un par de nodos, con dirección y además formarse al menos una figura cerrada dentro del grafo.

Un ejemplo que representa este tipo de grafo sería la representación del flujo que siguen los pacientes una vez que asisten a la consulta de cuerpo de guardia del ISSSTE, en la que las operaciones del proceso serían los nodos y la clasificación del tipo de paciente y su recorrido dentro de la instalación las aristas. En este caso el cliente al llegar pasa por la consulta de clasificación y ahí le asignan un color en función de la gravedad de su dolencia, se asumen tres colores, rojo, amarillo y verde, que van de mayor gravedad a menor respectivamente y de esta dependerá el tiempo de espera para pasar a la siguiente consulta donde se encuentra el doctor, existe tres consultas con doctores disponibles y de ahí pueden pasar al laboratorio, u a otras de las salas. Luego, del laboratorio pueden pasar nuevamente a la consulta del doctor para revisar los resultados y ocurriría un ciclo, o del doctor directo a la sala, luego al laboratorio y retornar nuevamente al doctor, y ocurriría otro ciclo. El algoritmo de acomodo que mejor se ajustó fue el *spring layout*. Este grafo se muestra en la figura 11 en la página 24 donde se muestra la representación gráfica del mismo.

```
import matplotlib.pyplot as plt
import networkx as nx

G=nx.MultiDiGraph()

G.add_node(1)
G.add_node(2)
G.add_node(3)
G.add_node(4)
G.add_node(5)

G.add_edge(1,2, weight=3)
G.add_edge(1,2, weight=4)
G.add_edge(1,2, weight=5)
G.add_edge(2,3, weight=3)
G.add_edge(2,3, weight=4)
G.add_edge(2,4, weight=3)

G.add_edge(2,5, weight=3)
G.add_edge(3,2, weight=4)
G.add_edge(3,2, weight=3)

y=[(1,2),(2,3),(3,2)]
g=[(1,2),(2,3),(3,2)]
r=[(1,2),(2,4),(2,5)]

pos= nx.spring_layout(G, k=1, iterations=50, threshold=0.0001, weight='weight',
    scale=1)

nx.draw_networkx_nodes(G, pos, node_size=400, node_color='b', node_shape='o')

nx.draw_networkx_edges(G, pos, edgelist=y, width=10, alpha=0.2,
edge_color='y', style='dashed')

nx.draw_networkx_edges(G, pos, edgelist=g, width=6, alpha=0.5,
edge_color='g')

nx.draw_networkx_edges(G, pos, edgelist=r, width=2, alpha=0.8,
edge_color='r')

labels = {}
labels[1] = r'1'
labels[2] = r'2'
labels[3] = r'Lab'
labels[4] = r'3'
labels[5] = r'4'


plt.axis('off')

nx.draw_networkx_labels(G, pos, labels, font_size=10)

plt.savefig("imagenes1/Fig11.eps")
plt.show()
```

grafo11lyspring.py

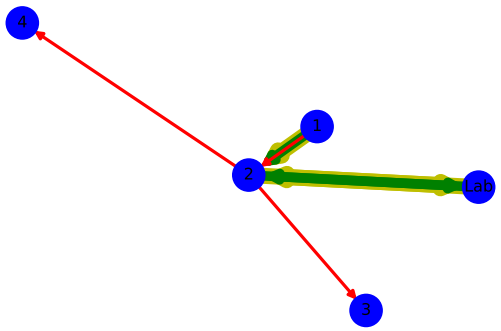

Figura 11: Recorrido del paciente en la sala de urgencias del ISSSTE

12. Multigrafo dirigido reflexivo usando *shell layout*

En los multigrafos dirigidos debe existir más de una arista entre un par de nodos, con dirección y además cada nodo debe llamarse a sí mismo.

Un ejemplo en el que el empleo de la teoría de grafos como los que se analizan en esta sección puede emplearse es en el estudio de la influencia de la religión en la sociedad, como un subejemplo de la aplicación en redes sociales.[2]. Por ejemplo, si cada nodo es una persona religiosa creyente, ese nodo (persona) constantemente se realiza un autoexamen de conciencia a sí mismo, evaluando cómo ha sido su actitud con respecto a las variables a analizar en este autoexamen. Estas variables pueden ser operacionalizadas como cada una de las doctrinas de cada religión, las cuales varían de una religión a otra, pero son más de una y pueden agruparse por categorías. Estas variables representan las aristas, cada una puede tener un peso determinado según sea el interés del estudio, a su vez estas mismas doctrinas (convertidas en variable) serán las que cada persona creyente se encargará de transmitir a otras personas creyentes o no, pudiera decirse que se transmite el tipo de actitud que se debe tener ante cada una de estas variables de una persona a otra. En este ejemplo se puede evaluar el impacto en un grupo poblacional que puede tener un tipo de creencia u otro en la medida en que aumente la red. Para este ejemplo se reservó el algoritmo de acomodamiento *shell layout* es un algoritmo que acomoda los nodos en círculos concéntricos [4] este transmite la idea de expansión que es justamente lo que se desea reflejar en el ejemplo propuesto, si se llevara a gran escala. Una simplificación de este tipo de grafo se representa en la figura 12 en la página 26.

```
import matplotlib.pyplot as plt
import networkx as nx

G=nx.MultiDiGraph()

G.add_node(1)
G.add_node(2)
G.add_node(3)
G.add_node(4)
G.add_node(5)

G.add_edge(1,2, weight=3)
G.add_edge(1,2, weight=4)
G.add_edge(1,2, weight=5)
G.add_edge(2,3, weight=3)
G.add_edge(2,3, weight=4)
G.add_edge(2,3, weight=4)
G.add_edge(3,4, weight=3)
G.add_edge(3,4, weight=3)
G.add_edge(3,4, weight=3)
G.add_edge(4,5, weight=3)
G.add_edge(4,5, weight=3)
G.add_edge(4,5, weight=3)

y=[(1,2),(2,3),(3,4),(4,5)]
g=[(1,2),(2,3),(3,4),(4,5)]
r=[(1,2),(2,3),(3,4),(4,5)]

pos=nx.shell_layout( G , nlist = None , scale = 1 , center = None , dim = 2 )
list=[]
list=G.nodes()
listacolor=['black', 'y', 'b','r','g']

for i in list:
    nx.draw_networkx_nodes(G, pos, node=i, node_size=300, node_color=listacolor,
    node_shape='o')
nx.draw_networkx_edges(G, pos, edgelist=y, width=8, alpha=1,
edge_color='y', style='dashed')
nx.draw_networkx_edges(G, pos, edgelist=g, width=6, alpha=0.5,
edge_color='g')
nx.draw_networkx_edges(G, pos, edgelist=r, width=2, alpha=0.8,
edge_color='r')
plt.axis('off')
plt.savefig("imagenes1/Fig13.eps")
plt.show()
```

12rectificado.py

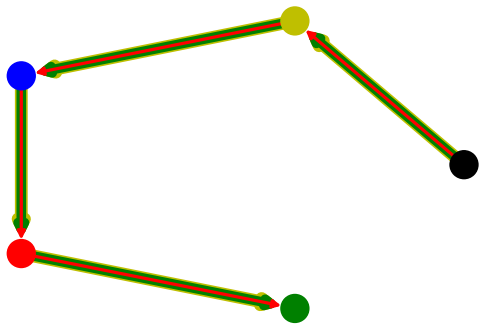

Figura 12: Expansión de doctrinas religiosas

Referencias

[1] Pieter Swart Aric Hagberg, Dan Schult. *NetworkX Reference, Release 2.3rc1.dev20190113142952*, 2019. URL `https://networkx.github.io/documentation/stable/reference/index.html`.

[2] Anwesha Chakraborty, Trina Dutta, Sushmita Mondal, and Asoke Nath. Application of graph theory in social media. *International Journal of Computer Sciences and Enginieering*, 6:722–729, 10 2018. doi: 10.26438/ijcse/v6i10.722729.

[3] Aric Hagberg. Forum de networkx. `http://https://groups.google.com/forum/#!topic/networkx-discuss/qhHjh0CGP8A`. Accessed: 2019-02-25.

[4] Desarrolladores NetworkX. `https://networkx.github.io/documentation/stable/reference/drawing.html`. Accessed: 2019-02-23.

[5] Elisa Schaeffer. Complejidad computacional de problemas y el análisis y diseño de algoritmos, 2017. URL `https://elisa.dyndns-web.com/teaching/aa/pdf/aa.pdf`.