Tarea 2

Rogelio Jesús Corrales Díaz

29 de enero 2019

1. Introducción

Este informe pretende realizar un análisis de las distintas formas de posicionamiento que ofrece la librería NetworkX. Es importante destacar el papel que juega la representación en teoría de grafos. Una correcta representación puede facilitar el análisis del problema.

La librería NetworkX contiene varios algoritmos de ordenamiento estos siguen distintos principios de posicionamiento de los nodos. Precisamente por estas diferencias es que dichos métodos no resultan útiles para todos los problemas que se pretenden representar con teoría de grafos.

A continuación se presentara un comparación, los grafos generados sin especificar métodos de ordenamiento serán obtenidos ahora usando el layout mas conveniente en cada caso.

Como parte de este breve preámbulo se enumeran los tipos de ordenamientos utilizados. $random//spring//spectral//circular//pipartite//kamada_kawai//shell$

2. Grafo simple no dirigido acíclico

En la figura 1 de la página 2 se muestra un grafo que representa un problema de un árbol de expansión. Es evidente que no existen ciclo ya que no hay forma de partir de un nodo y escoger un recorrido que regrese al origen. Este grafo puede representar un árbol de expansión de una red eléctrica donde el nodo origen es el número 1 y este va alimentando al dos y al tres que a su vez alimentan a otros.

Para la representación de este grafo se utilizaron tres algoritmos de ordenamiento con el objetivo de hacer una comparación. En el primer las posiciones de los nodos fueron generadas de forma aleatoria 1 de la página 2, mientras que en el

segundo fueron posicionados usando el *circular*$_l$*ayout* 2 de la página 3. Como es posible observar en estos dos ejemplos la forma en que la red está representada no la comprención del problema sobre todo por los entrecruzamiento entre los vértices.

Es por esto que se hace una misma representación con *spring*$_l$*ayout* 3 de la página 3. En esta ocación se solucionan los problemas de entrecruzamiento. Este método se basa en encontrar una distribución tal que los nodos tienen una carga y el objetivo es que estos se encuentren interactuando según su carga que define el acomodo de los nodos.

```
import networkx as nx
import matplotlib.pyplot as plt
G=nx.Graph()
G.add_edges_from([(1,2),(1,3), (3,4),(3,5),(4,10),(5,8),(2,6),
    (6,11), (2,7), (7,12)])
nx.draw(G, with_labels=True, pos=nx.spring_layout(G), edge_color='
    black', node_color='gray',node_size=1000, edgecolors='black',
    font_weight='bold')
G.number_of_nodes()
plt.savefig('grafoNAS.eps', format='eps', dpi=1000)
```

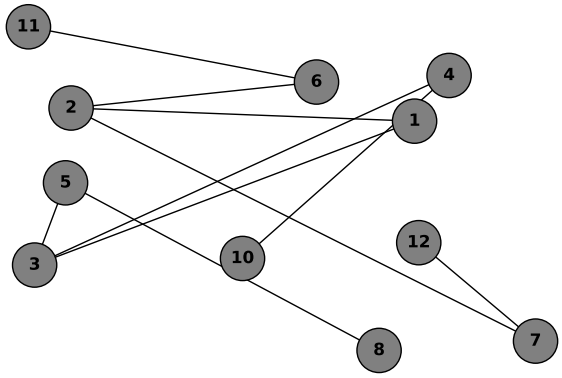

Figura 1: Grafo no dirigido acíclico

3. Grafo simple no dirigido cíclico

La figura 4 de la página 4 representa un grafo que puede ajustarse a un problema ampliamente conocido y estudiado, el problema del agente viajero. Este problema consiste en recorrer cada uno de los nodos una vez, minimizando la

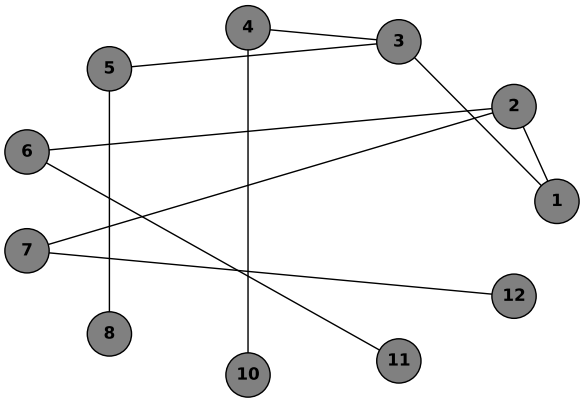

Figura 2: Grafo no dirigido acíclico

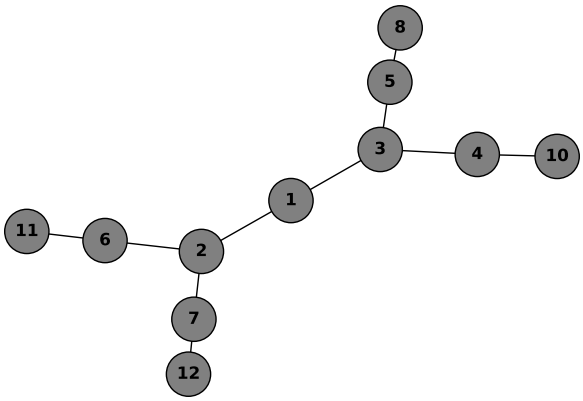

Figura 3: Grafo no dirigido acíclico

distancia total. Este es resulta un problema de optimización combinatoria y esta comprendido dentro del grupo NP duro. Esto muestra a lo que hacíamos referencia cuando se enunció que la existencia de ciclos complejiza el problema. Ya que en problemas como estos aunque el numero de soluciones factibles son finitas con el crecimiento del numero de nodos se hace improbable encontrar un algoritmo que arroje una solución en tiempo polinomial.

Este es n ejemplo en el cual funcional utilizar $kamada_kawai_layout$ 5 de la página 4 ya que si se realiza una comparación entre este y la figura anterior puede ser concluido como se eliminan buena parte de los entrecruzamientos, y es que este algoritmo precisamente persigue este objetivo.

```
1 G=nx.Graph()
```

```
2 G.add_edges_from([(1,2),(2,3), (3,4),(4,5),(5,6),(6,7),(7,8), (8,1)
      ])
```

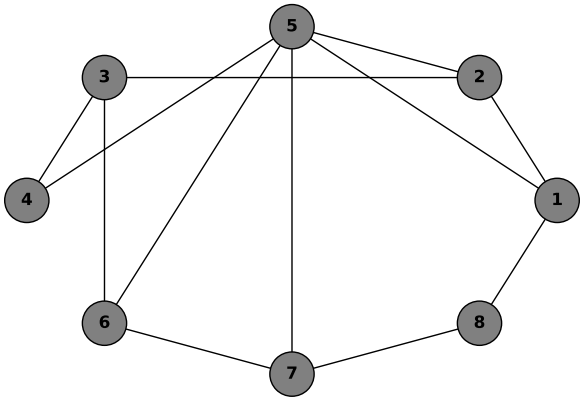

Figura 4: Grafo no dirigido cíclico

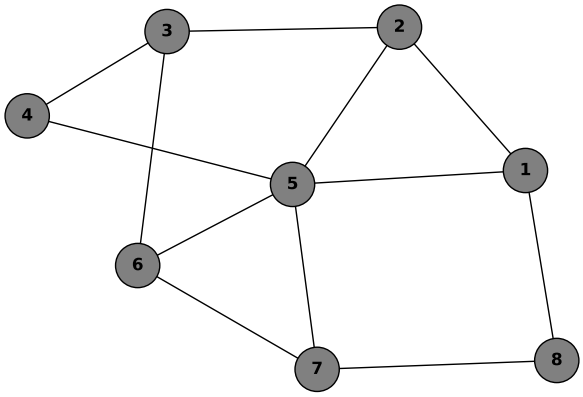

Figura 5: Grafo no dirigido acíclico

4. Grafo simple no dirigido reflexivo

En este caso se representan los nodos reflexivos con el color rojo, este grafo podría representar un grupo de tareas que se complementan entre si y no importa el orden en que se realicen. Además las tareas representadas por los nodos reflexivos cuentan con un reproceso en caso de que sea necesario garantizar mayor calidad en el producto final figura 6 en la página 5.

```
import networkx as nx
import matplotlib.pyplot as plt
G=nx.Graph()
G.add_edges_from([(1,2), (1,1), (2,3), (3,4), (4,1), (1,3), (2,4)])
node1 = {1,2}
node2 = {3,4}
pos = {1:(200, 350), 2:(550,350), 3:(650, 220), 4:(400,100),
    5:(150,220)}
nodes=nx.draw_networkx_nodes(G, pos, with_labels=True, nodelist=
    node1,node_size=400, node_color='r', node_shape='o')
nodes=nx.draw_networkx_nodes(G, pos, with_labels=True, nodelist=
    node2,node_size=400, node_color='grey', node_shape='o')
nodes=nx.draw_networkx_edges(G, pos)
nx.draw_networkx_labels(G, pos)
plt.savefig('tercero con numeros.eps', format='eps', dpi=1000)
```

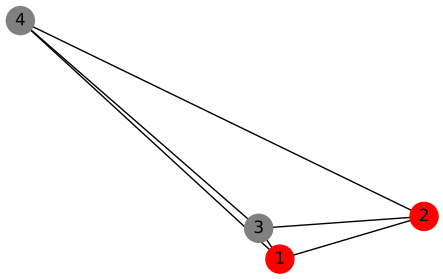

Figura 6: Grafo no dirigido reflexivo

5. Grafo simple dirigido acíclico

Con este tipo de grafos pueden ser representados redes de distribución eléctrica comenzando por un nodo principal que alimenta a un grupo de nodos secundarios que a su vez alimentan a otros figura 7 página 6.

Para la representación en este caso fueron utilizados $circular_layout$,$random_layout$ y $shell_layuot$; Este último funciona ordenando los nodos en circunferencias concéntricas de distintos radios haciendo capas. Este algoritmo no elimina los entrecruzamientos.

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

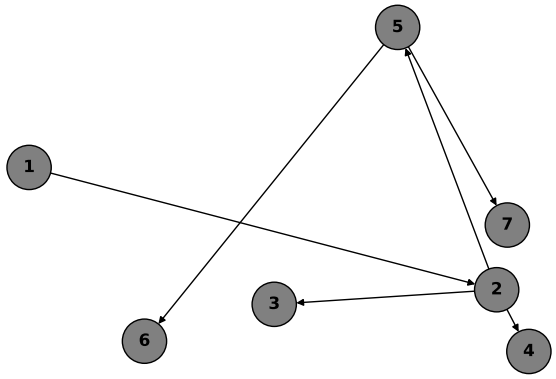

Figura 7: Grafo dirigido acíclico

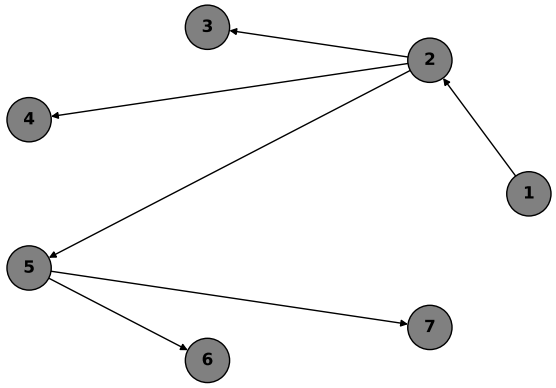

Figura 8: Grafo dirigido acíclico

6. Grafo simple dirigido cíclico

El grafo mostrado en la figura 10 página 7 pudiera corresponder a una red de rutas de una ciudad donde tomando la primera ruta sería posible llegar a las rutas 2 y 3 y de esta forma como expresa la figura. Como podemos observa los vértices se encuentran unidos por aristas que tienen una dirección esto hace se reduzcan las variantes de mover flujo por el grafo.

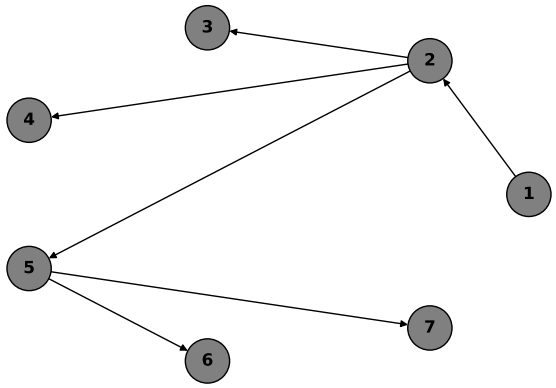

Figura 9: Grafo dirigido acíclico

```
G=nx.DiGraph()
G.add_edges_from([(1,2),(2,3), (3,4), (1,3), (4,1)])
```

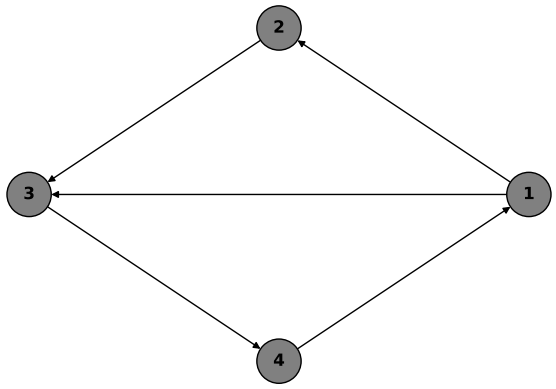

Figura 10: Grafo no dirigido acíclico

7. Grafo simple dirigido reflexivo

Imaginemos que tenemos una linea de producción con N procesos que quedan representados cada uno por un nodo, se conoce que los procesos 1 y 2 son obligatorios. Pero para obtener el producto deseado es necesario continuar entonces

debe ser tomada la decisión de continuar con el proceso 3 o el 6 y asi de esta manera hasta recorrer todos los nodos figura 11 página 8.

```
G=nx.DiGraph()
G.add_edges_from([(1,2),(2,3), (3,4)])
node1 = {1,2}
node2 = {3,4}
pos = {1:(200, 350), 2:(550,350), 3:(650, 220), 4:(400,100),
    5:(150,220)}
nodes=nx.draw_networkx_nodes(G, pos, with_labels=True, nodelist=
    node1,node_size=400, node_color='r', node_shape='o')
nodes=nx.draw_networkx_nodes(G, pos, with_labels=True, nodelist=
    node2,node_size=400, node_color='grey', node_shape='o')
nodes=nx.draw_networkx_edges(G, pos)
nx.draw_networkx_labels(G, pos)
plt.savefig('tercero con numeros.eps', format='eps', dpi=1000)
```

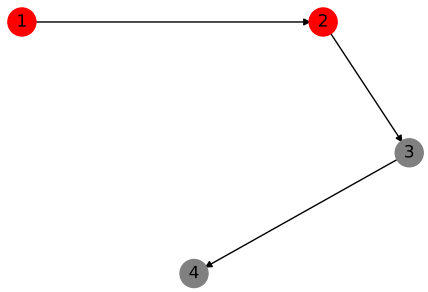

Figura 11: Grafo simple dirigido reflexivo

8. Multigrafo no dirigido acíclico

Pudiéramos interpretar este grafo figura 12 página 9 como la conexión entre calles, donde existen dos maneras distintas de unirlas a excepción del nodo 4.

```
G=nx.MultiGraph()
G.add_edges_from([(1,2), (2,1), (2,3,), (3,2,), (3,4,)])
```

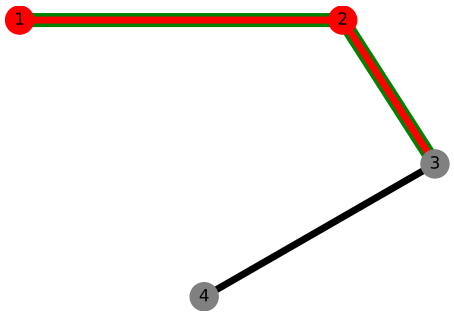

Figura 12: Multigrafo no dirigido acíclico

9. Multigrafo no dirigido cíclico

Imaginemos que tenemos representado cuatro procesos que generan información y que la información generada por cada uno debe ser compartida. Con este fin se conoce que de un proceso a otro (nodos) existen dos variantes de enviar la información. Ahora bien cada variante tiene un tiempo de transmición distinto. El objetivo definido por los decisores seráminimizar este tiempo. Este problema queda representado por el grafo del la figura 13 en la pagina 10. Donde los nodos representan cada uno de los procesos y las aristas las variantes.

```
G=nx.MultiGraph()
G.add_edges_from([(1,2), (2,1), (2,3,), (3,2,), (3,4,), (4,1),
    (1,4),(4,2),(2,4),(3,1),(1,3)])
```

10. Multigrafo no dirigido reflexivo

Ahora imaginemos que nos encontramos en una situación parecida que la sección anterior. Pero resulta que los procesos 1 y 2 necesitan la infromación que ellos mismos generan para realizar controles. Y anteriormente la mandaban a los otros procesos para que fuera procesada. Si se valorara la opción de que esta fuera estudiada en el mismo proceso con un tiempo de procesamiento determinado. Estaríamos en presencia del problema representado por el grafo de la figura 14 en la pagina 10.

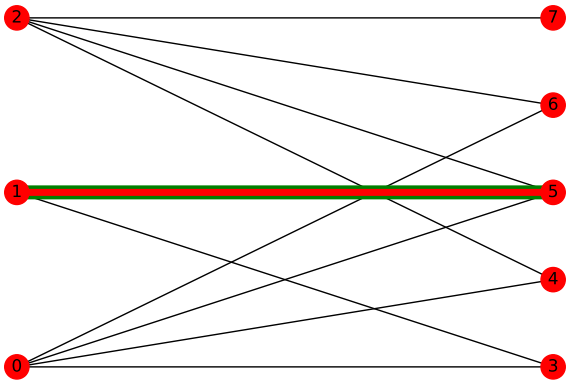

Figura 13: Multigrafo no dirigido cíclico

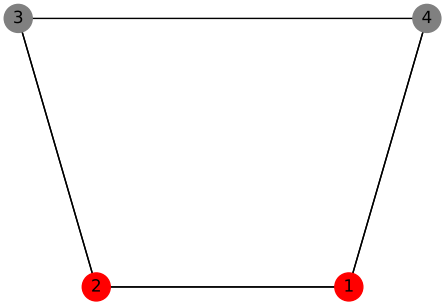

Figura 14: Multigrafo no dirigido acíclico

```
G=nx.MultiGraph()
G.add_edges_from([(1,2),(1,1),(2,1),(2,2),(2,3,),  (3,2,),  (3,4,),
      (4,1),  (1,4)])
```

11. Multigrafo dirigido acíclico

Supongamos que una empresa cuenta con 3 almacén desde el cual tiene que transportar las mercancías a dos clientes, pero antes esta debe pasar por controles de calidad para lo cual es transportada hacia la sede principal. Si sabemos

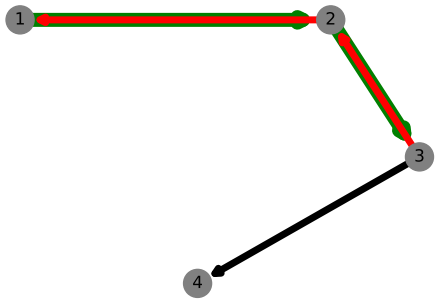

Figura 15: Multigrafo dirigido acíclico

que la capacidad de cada viaje es limitada y dependiente del número de vehículos. Nos encontramos con un problema de transporte que esta representado por el grafo de la figura 15 en la página 11. Donde los nodos A=[4,5,6] S=[2] C=[1,3] representan los almacenes, la empresa y los clientes respectivamente.

```
G=nx.MultiDiGraph()
G.add_edges_from([(1,2),(2,1),(2,3),(3,2),(4,2),(5,2), (6,2)])
```

12. Multigrafo dirigido cíclico

Retomemos ahora el problema del TSP, en este problema los nodos estan unidos por aristas que no necesariamente deben de estar dirigidas, ya que pueden ser usadas en ambas direcciones. Pero si a las posible le agregamos varias rutas que unen los destinos y a su vez estas tienen un solo destino, este problema es representado por un multigrafo dirigido cíclico figura 16 pagina 12.

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

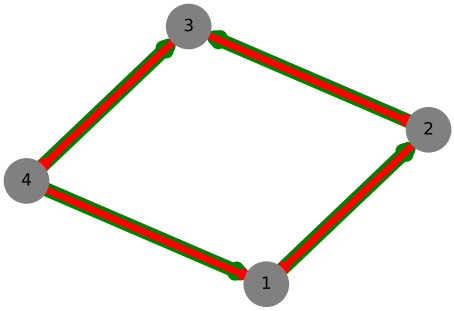

Figura 16: Multigrafo no dirigido cíclico

13. Multigrafo dirigido reflexivo

Supongamos que una linea de producción fabrica 4 componentes distintos pero solo puede ser producido un componente a la vez. Por esto cuando se desee fabricar otro hay que parar la producción y cambiar la configuración lo cual conlleva un tiempo de ejecución. Ademas linea deberá parar cada ciertos periodos de tiempo por mantenimientos preventivos, y cada vez que pare se deberá re-configurar ya que fue reiniciada. Este problema se puede plantear como un TSP pero además como existe la posibilidad de que luego del mantenimiento se vuelva a usar la misma configuración se vuelve un problema representado por un Multigrafo dirigido reflexivo como el de la figura 17 de la página 13.

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

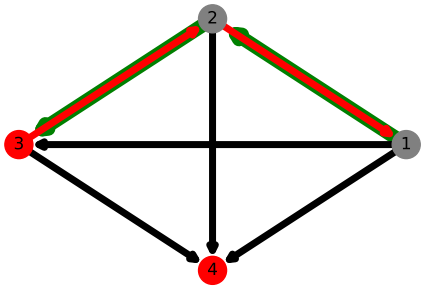

Figura 17: Multigrafo dirigido reflexivo