Prueba

Rogelio Jesús Corrales Díaz

29 de enero 2019

1. Introducción

Este reporte pretende realizar un análisis de los principales tipos de grafos así como sus aplicaciones, y características principales. Para esto comenzaremos definiendo que es un grafo:

Un grafo G se define como un par (V,E), donde V es un conjunto cuyos elementos son denominados vértices o nodos y E es un subconjunto de pares no ordenados de vértices y que reciben el nombre de aristas o arcos.

Existen distintos tipos de grafos cada uno con sus especificidades. Atendiendo a sus características pudieran clasificarse de forma mas general como grafos simples o multigrafos.

Un grafo simple es aquel en el que para cada nodo U existe solo una arista que lo conecta con otro nodo V. Mientras que en un multigrafo dos vértices pueden esta conectados por mas de una arista.

Otra de las clasificación entre los grafos que resulta muy importante destacar,

es si hay presencia de ciclos. Un grafo es acíclico cuando no es posible encontrar un camino tal que si comenzamos el recorrido en el nodo U (origen) el camino escogido nos regrese al dicho origen.

Esta ultima característica es bien importante ya que de aparecer ciclos en una grafo, encontrar la solución del problema que representa se complejiza.

Grafos dirigidos son aquellos donde la arista A que une dos vértices U y V se encuentra orientada en un sentido fijo de manera tal que el flujo que pasa por el vértice solo puede ir en el sentido previamente definido. En caso de que dicha dirección no sea estricta y los nodos sean unidos por una sola arista el grafo se clasifica como no dirigido.

Por último un grafo reflexivo es aquel donde existe una arista que conecta un vértice con si mismo.

En el documento será mostrado el código de Python para generar cada uno de los grafos explicados. Fue utilizada la librería Networkx. De esta librería fueron usadas funciones para generar los grafos: Graf para los simples, MutiGraf para los multigrafos, Digraf para los grafos dirigidos y MultiDigraf para los multigrafos dirigidos. Además los nodos reflexivos fueron resaltado en color rojo.

2. Grafo simple no dirigido acíclico

En la figura 1 de la página 3 se muestra un grafo que representa un problema de un arbol de expancion. Es evidente que no existen ciclo ya que no hay forma de partir de un nodo y escoger un recorrido (Conjunto de aristas) que nos regrese al origen. Este grafo puede representar un arbol de expancion de una red eléctrica. Donde el nodo origen es el numero 1 y este va alimentando al dos y al tres que a su vez alimentan a otros.

```
1 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)
```

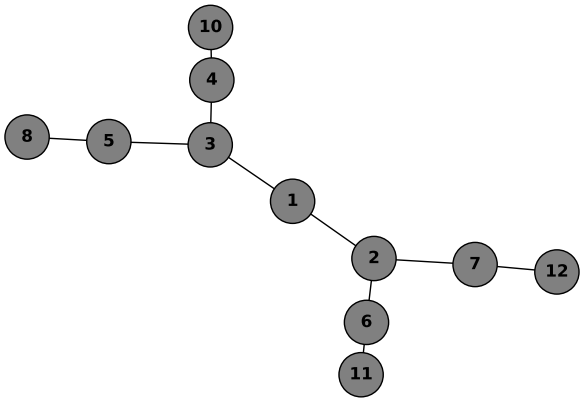

Figura 1: Grafo no dirigido acíclico

3. Grafo simple no dirigido cíclico

La figura 2 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 distancia total. Este es resulta un problema de optimización combinatoria y esta comprendido dentro del grupo NP hard. 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.

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

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

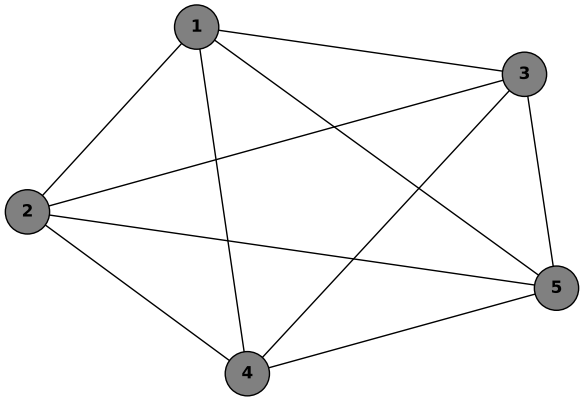

Figura 2: Grafo no dirigido cíclico

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 ademas 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 3 en la pagina 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)
```

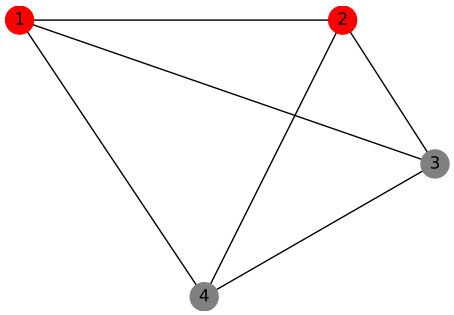

Figura 3: Grafo no dirigido reflexivo

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 4 página 6

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

6. Grafo simple dirigido cíclico

El grafo mostrado en la figura 5 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

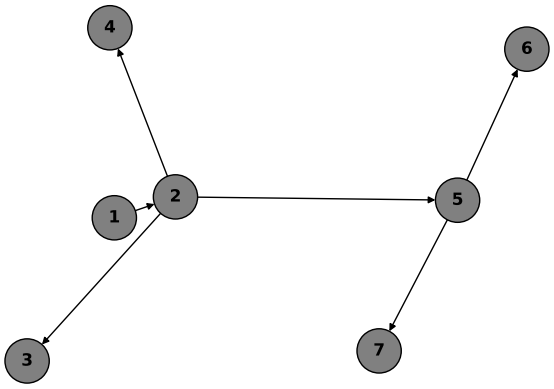

Figura 4: Grafo dirigido acíclico

vértices se encuentran unidos por aristas que tienen una dirección esto hace se reduzcan las variantes de mover flujo por el grafo.

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

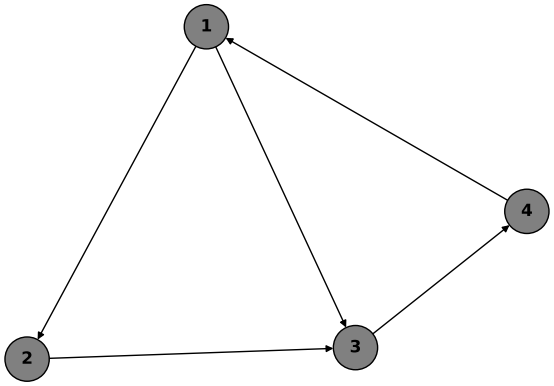

Figura 5: Grafo no dirigido acíclico

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 6 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)
```

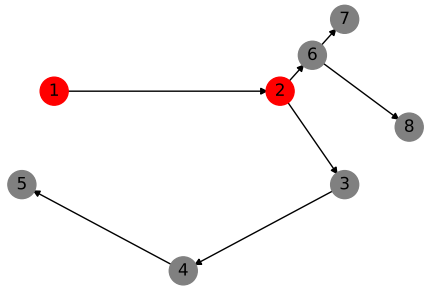

Figura 6: Grafo simple dirigido reflexivo

8. Multigrafo no dirigido acíclico

Pudiéramos interpretar este grafo figura 7 página 8 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,)])
```

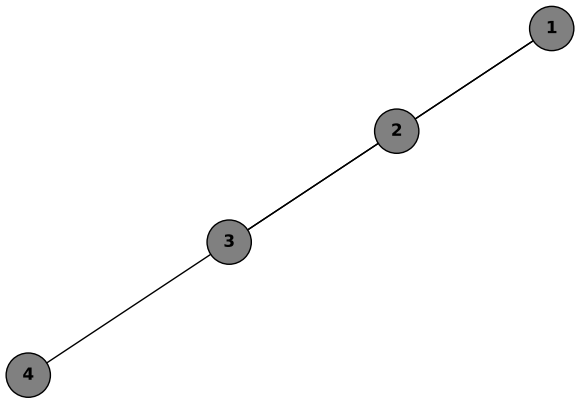

Figura 7: Multigrafo no dirigido acíclico

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 8 en la pagina 10. Donde los nodos representan cada uno de los procesos y las aristas las variantes.

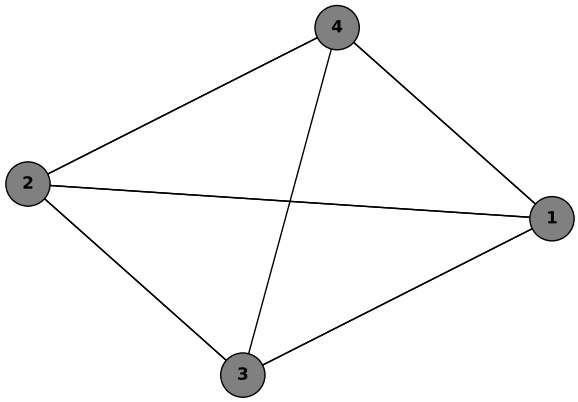

Figura 8: Multigrafo no dirigido cíclico

```
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 9 en la pagina 10.

```
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)])
```

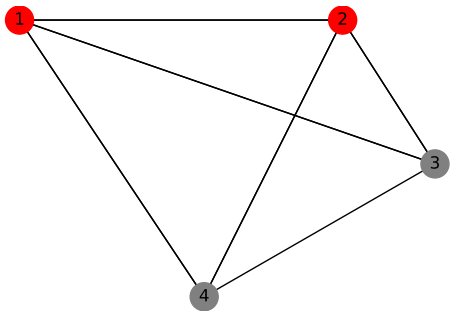

Figura 9: Multigrafo no dirigido acíclico

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 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 10 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 11 pagina 12.

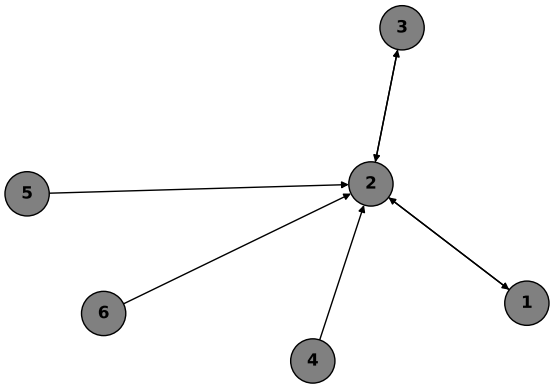

Figura 10: Multigrafo dirigido acíclico

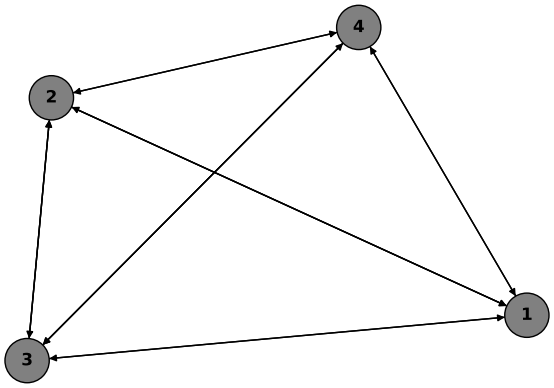

Figura 11: Multigrafo no dirigido cíclico

```
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)])
```

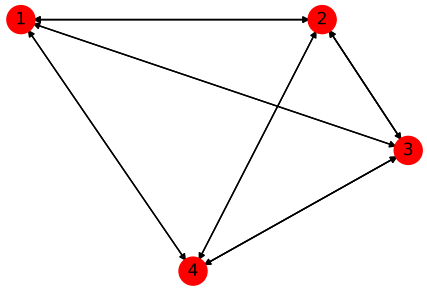

Figura 12: Multigrafo dirigido reflexivo

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 12 de la página 13.

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