Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Flujo en redes

PORTAFOLIO

Rogelio Jesús Corrales Díaz
1985269
Semestre: Enero-Junio 2019
Dra. Elisa Schaeffer

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 estar conectados por más 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 última 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: Graph para los simples, MutiGraph para los multigrafos, DiGraph para los grafos dirigidos y MultiDiGraph 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)
```

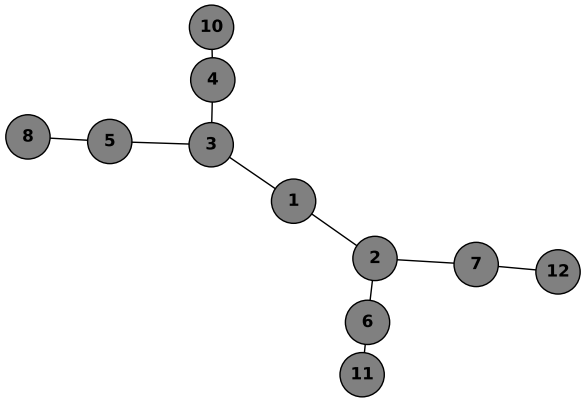


Figura 1: Grafo no dirigido acíclico

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)
    ])
```

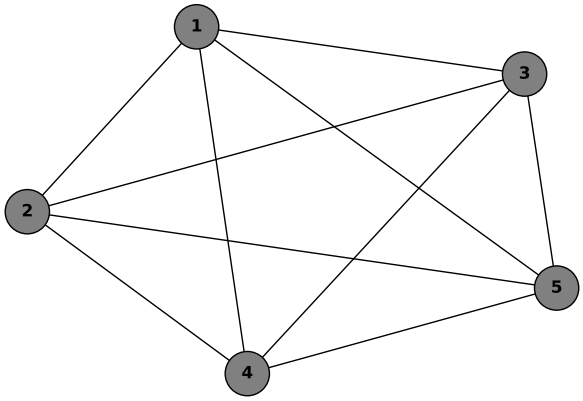


Figura 2: Grafo no dirigido cíclico

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)
```

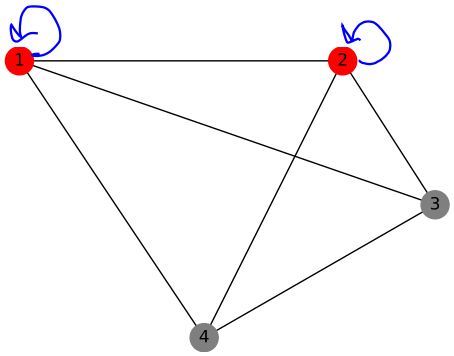


Figura 3: Grafo no dirigido reflexivo

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

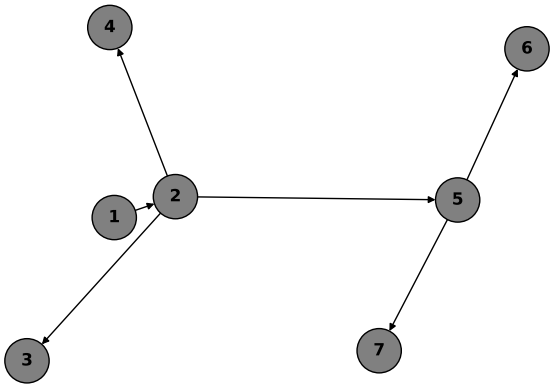


Figura 4: Grafo dirigido acíclico

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)])
```

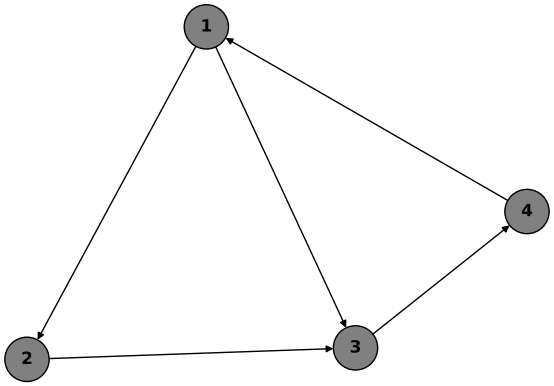


Figura 5: Grafo no dirigido acíclico

7. Grafo simple dirigido reflexivo

Imaginemos que tenemos una línea 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 así 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)
```

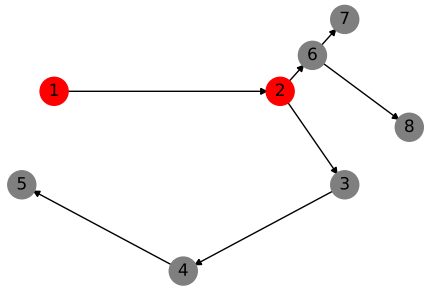


Figura 6: Grafo simple dirigido reflexivo

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,)])
```

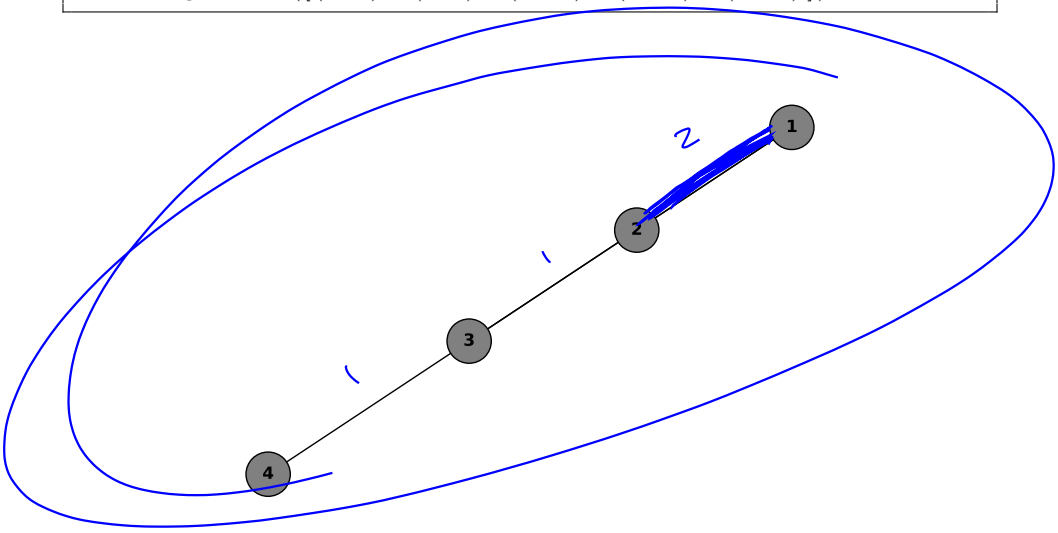


Figura 7: Multigrafo no dirigido acíclico

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 transmisión distinto. El objetivo definido por los decisores será minimizar 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.

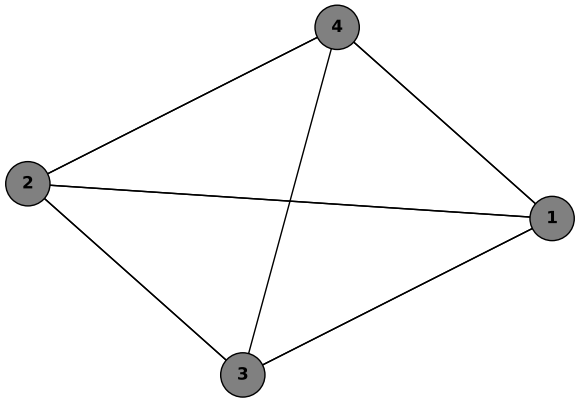


Figura 8: Multigrafo no dirigido cíclico

```
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)])
```

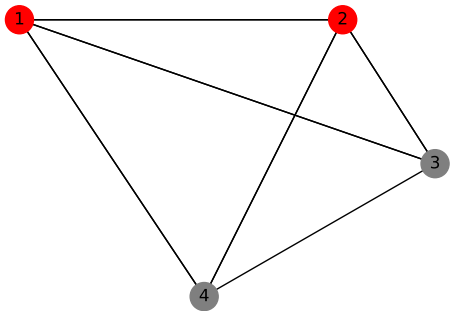


Figura 9: Multigrafo no dirigido acíclico

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 está 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.

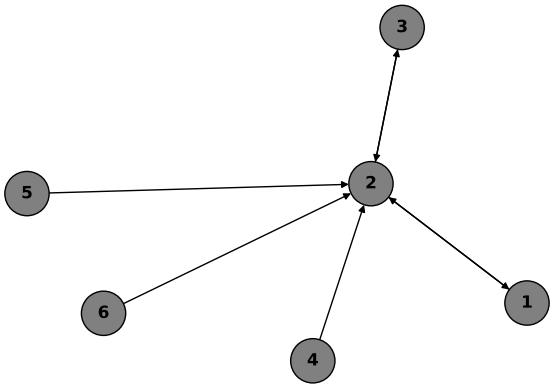


Figura 10: Multigrafo dirigido acíclico

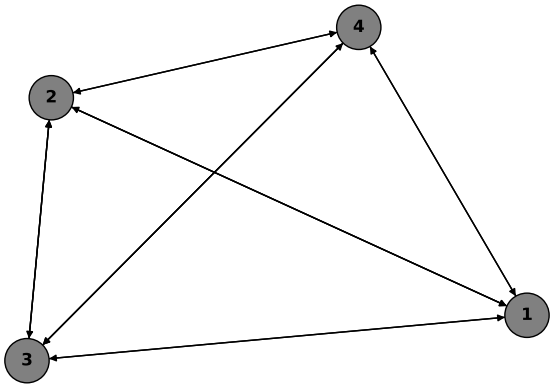


Figura 11: Multigrafo no dirigido cíclico

```
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)])
```

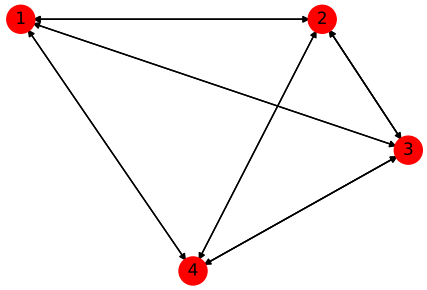


Figura 12: Multigrafo dirigido reflexivo

13. Multigrafo dirigido reflexivo

Supongamos que una línea 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. Además línea 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)])
```

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 más 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_layout* 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_layout* 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)
```

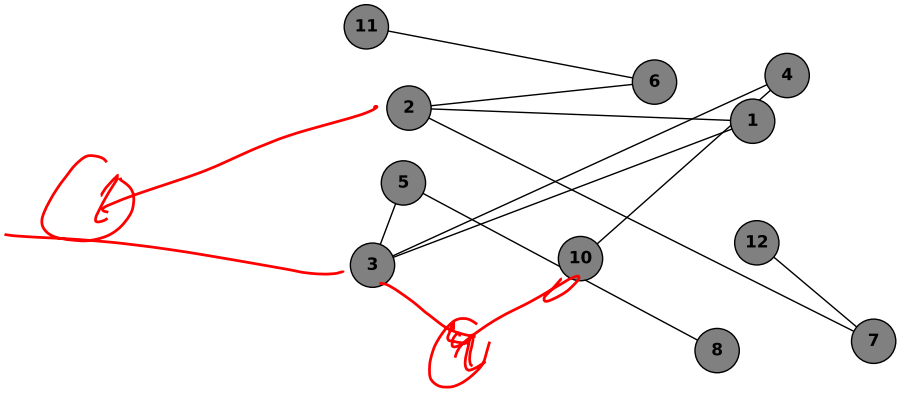


Figura 1: Grafo no dirigido acíclico

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

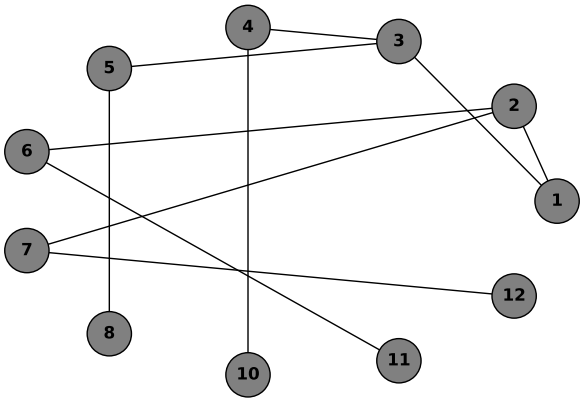


Figura 2: Grafo no dirigido acíclico

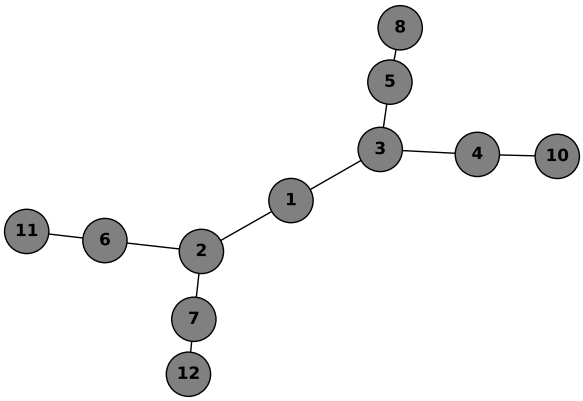


Figura 3: Grafo no dirigido acíclico

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)
      ])
```

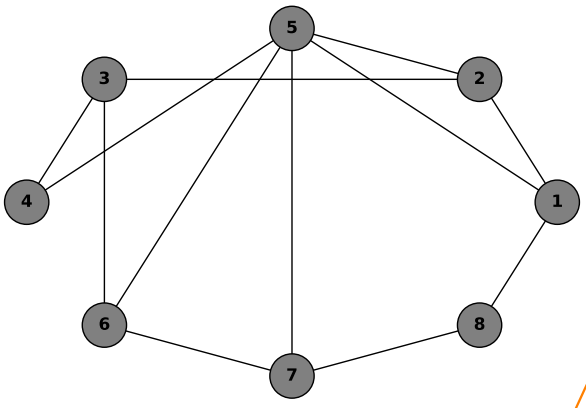


Figura 4: Grafo no dirigido cíclico

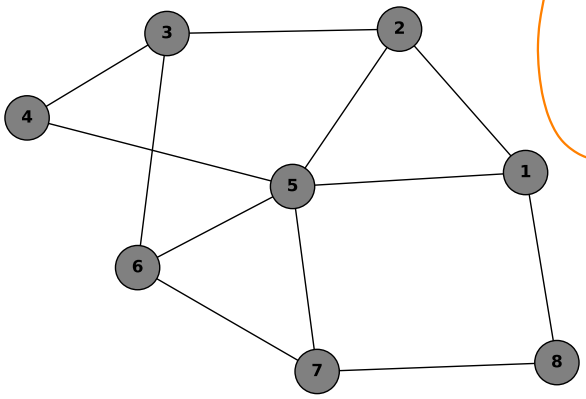


Figura 5: Grafo no dirigido acíclico

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)
```

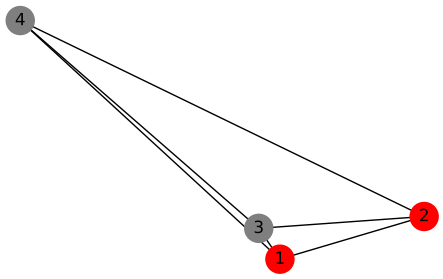


Figura 6: Grafo no dirigido reflexivo

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*$_l$*ayout*,*random*$_l$*ayout* y *shell*$_l$*ayuot*; 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)])
```

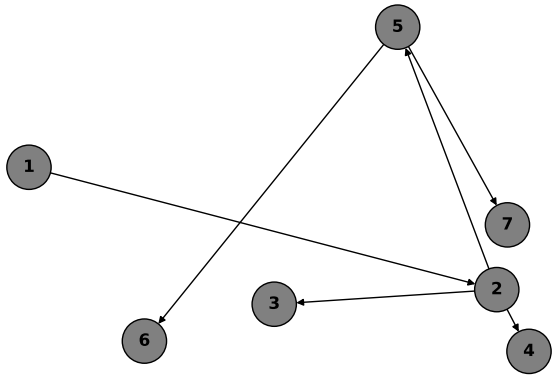


Figura 7: Grafo dirigido acíclico

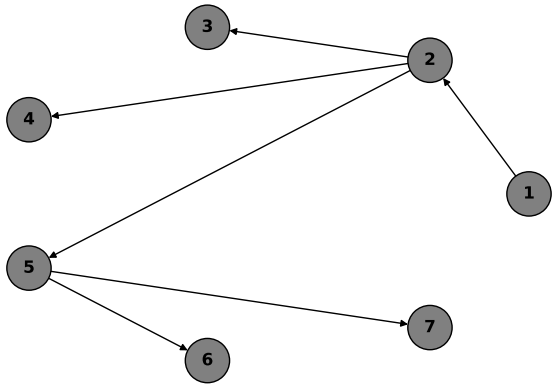


Figura 8: Grafo dirigido acíclico

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.

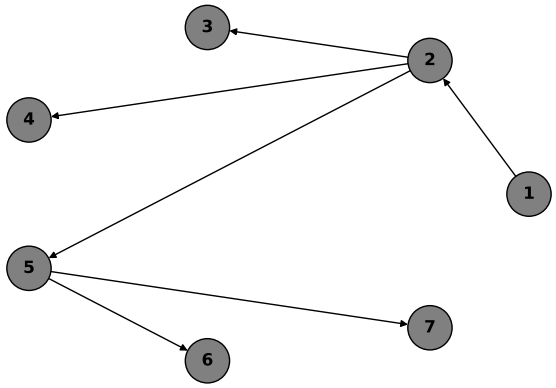


Figura 9: Grafo dirigido acíclico

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

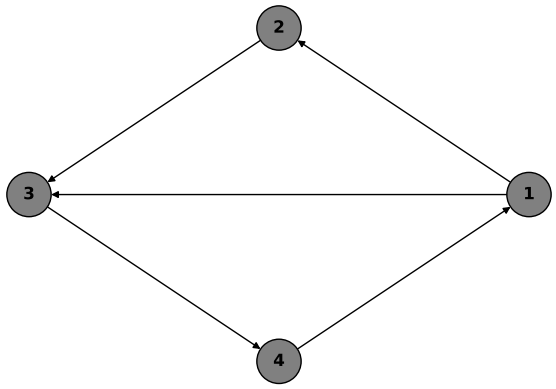


Figura 10: Grafo no dirigido acíclico

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 así 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)
```

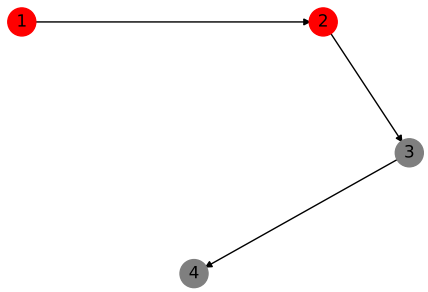


Figura 11: Grafo simple dirigido reflexivo

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,)])
```

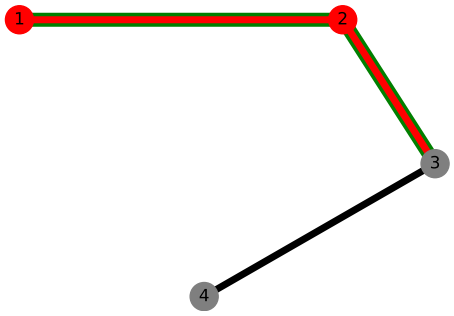


Figura 12: Multigrafo no dirigido acíclico

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.

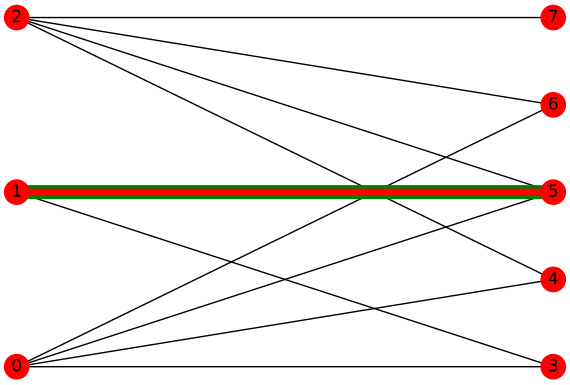


Figura 13: Multigrafo no dirigido cíclico

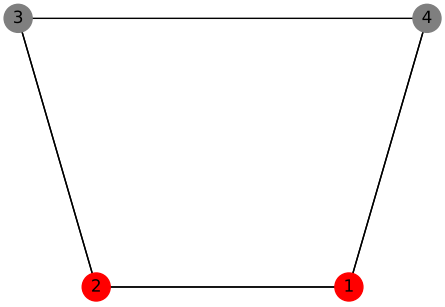


Figura 14: Multigrafo no dirigido acíclico

```
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

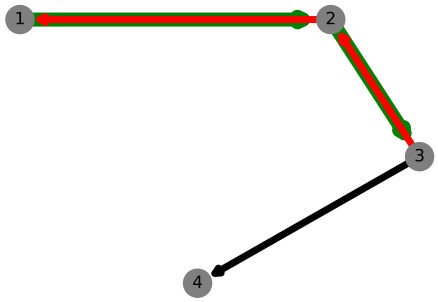


Figura 15: Multigrafo dirigido acíclico

que la capacidad de cada viaje es limitada y dependiente del número de vehículos. Nos encontramos con un problema de transporte que está 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)]))
```

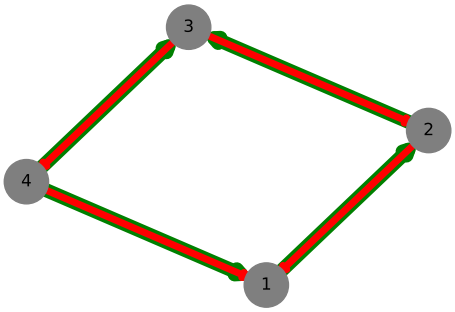


Figura 16: Multigrafo no dirigido cíclico

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.

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

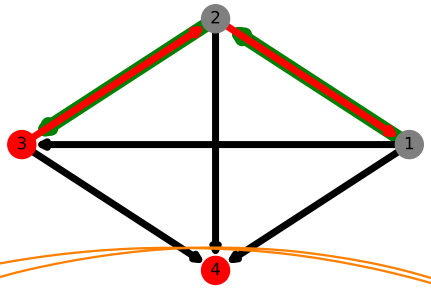


Figura 17: Multigrafo dirigido reflexivo

Tarea 3

1985269

18 de marzo de 2019

1. Resumen

Fueron escogidos cinco algoritmos de la librería NetworkX, cada uno de estos resolvió cinco problemas generados de forma aleatoria. Fueron realizadas 30 replicas durante un segundo para cada problema con cada algoritmo. A continuación se muestra la información recolectada a partir del análisis de los tiempos de procesamiento.

2. Deep first search tree

En los datos figura 1,2,3 pueden ser obsevardos la densidad con la que se distribuyen los tiempos medios que demora este algoritmo para resolver cuatro de los cinco problemas generados. Cada uno las gráficas muestran que el comportamiento de los datos no sigue un distribución normal y pesentan datos atípicos y saltos en los intervalos de frecuencia.

A continuación se comparte el código de Python con el que se recopilación la información

```
Graf=nx.Graph()
rog=0
diccionario_inst_Shortest_path={}
while rog<=4:
    Graf.clear()
    rango=random.randint(random.randint(10,len(matrizadd)),len(matrizadd))
    for i in range(rango):
        for j in range(rango):
            if matrizadd[i,j]!=0:
                matrizadd[j,i]=0
                #Graf.add_weighted_edges_from([(i,j,matrizadd[i,j])])
                Graf.add_edges_from([(i,j)])
    cont=0
    lista_tiempos=[]
    lista_tiempos_completos={}
    tiempo_de_paro=0
    tiempo_inicial=0
    tiempo_final =0
    tiempo_ejecucion=0
    for r in range(30):
        lista_tiempos_completos[r+1]=[]
        tiempo_de_paro=0
        while tiempo_de_paro<1:
            tiempo_inicial = time()
            nx.shortest_path(Graf, source=None, target=None, weight=None,
    method='dijkstra')
            tiempo_final = time()
            tiempo_ejecucion = tiempo_final - tiempo_inicial
            if tiempo_ejecucion>0.0:
                lista_tiempos_completos[r+1].append((tiempo_ejecucion*10000))
            tiempo_de_paro+=tiempo_ejecucion
    guardar_n_e[rog]=[]
    diccionario_inst_Shortest_path[rog]=[]
    for i in lista_tiempos_completos.keys():
         media=np.mean(lista_tiempos_completos[i])
         diccionario_inst_Shortest_path[rog].append(media)
    guardar_n_e['nodos'].append(len(Graf.nodes))
    guardar_n_e['edges'].append(len(Graf.edges))
    guardar_n_e['media'].append(np.mean(diccionario_inst_Shortest_path[rog]))
    guardar_n_e['desv'].append(np.std(diccionario_inst_Shortest_path[rog]))
    rog+=1
```

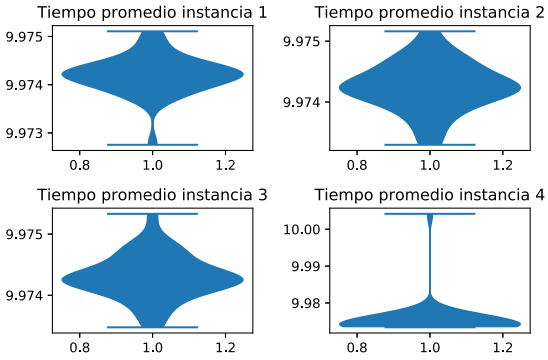


Figura 1: Violinplot dfstree

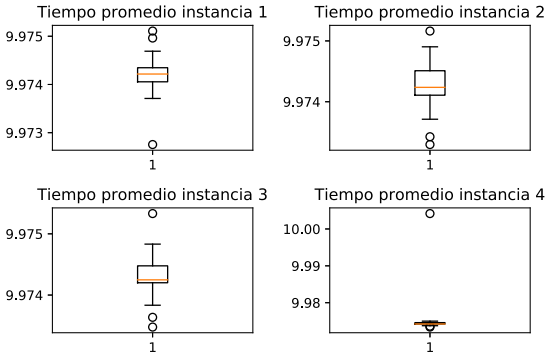


Figura 2: Boxplot dfstree

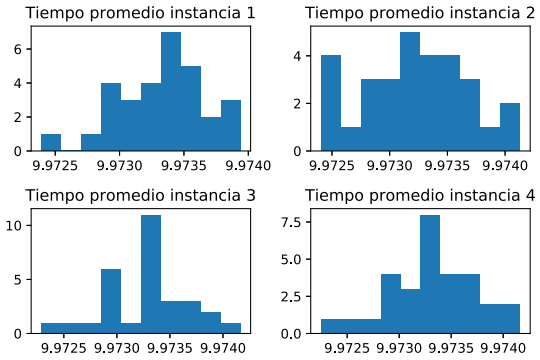


Figura 3: dfstree

3. Greedy color

En este caso la mayoría de los datos figuras 4,5,6 se encuentran en un intervalo con observaciones aisladas en otro intervalo mayor. Tampoco hay normalidad y existe un sesgo a la izquierda.

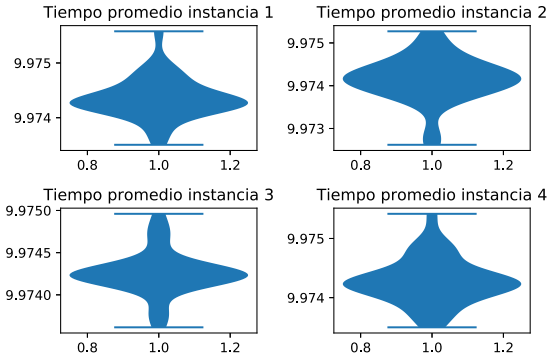


Figura 4: Violinplot greedy color

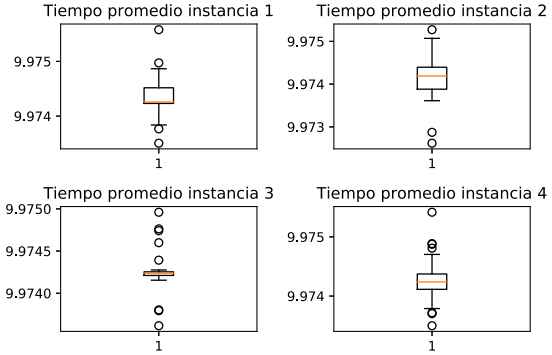


Figura 5: Boxplot greedy color

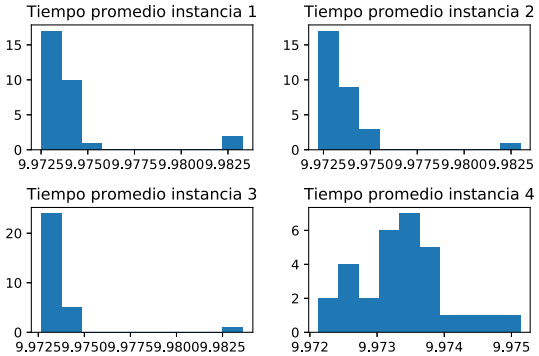


Figura 6: Greedy color

4. Maximun flow

De forma parecida las figuras muestran 7,8,9 como los datos se encuentran en intervalos separados y no siguen una distribución precisa, no se aprecia ninguna relación con las característica de la instancia.

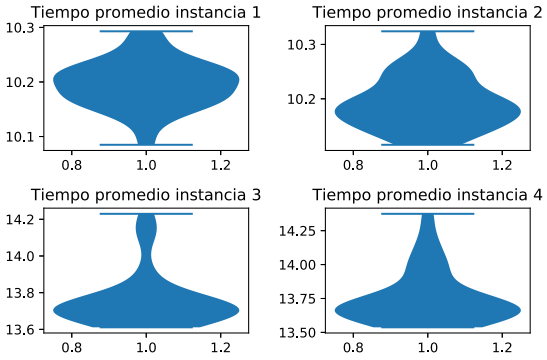


Figura 7: Violinplot maximun flow

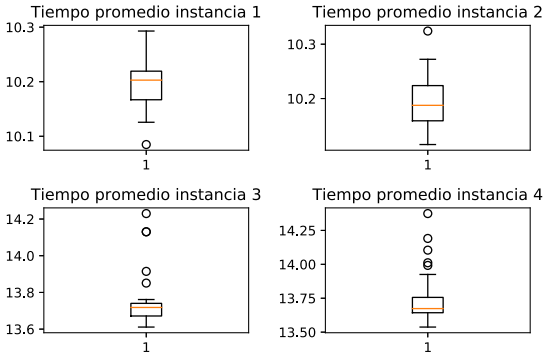


Figura 8: Boxplot maximun flow

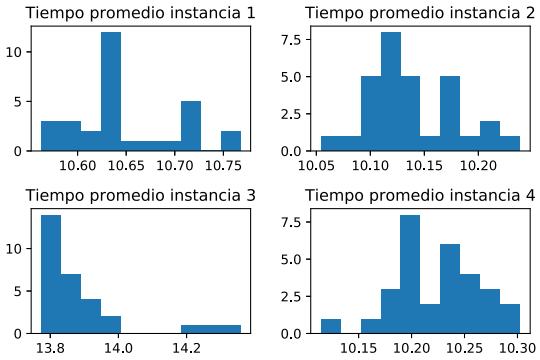


Figura 9: Maximun flow

5. Shortest path

En esta ocación vemos una mayor difenrencia entre los tiempos promedios de ejecucioón aun asi de estos no se infiere ningun comportamiento de probabilidad.

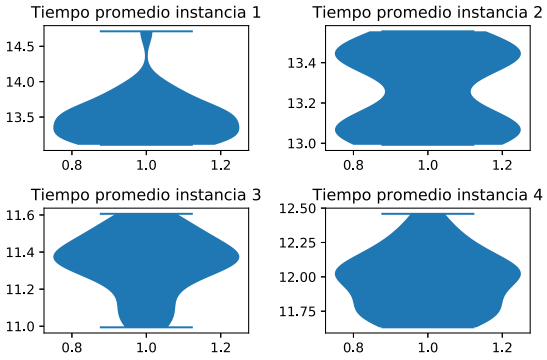


Figura 10: Violinplot Shortest path

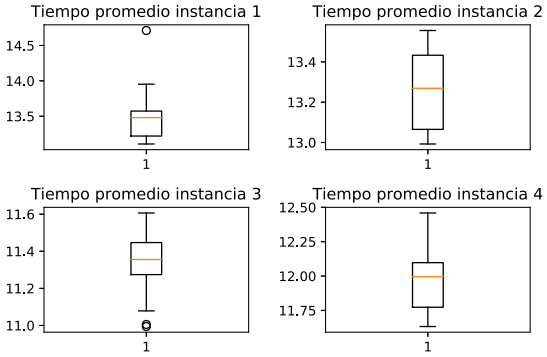


Figura 11: Boxplot Shortest path

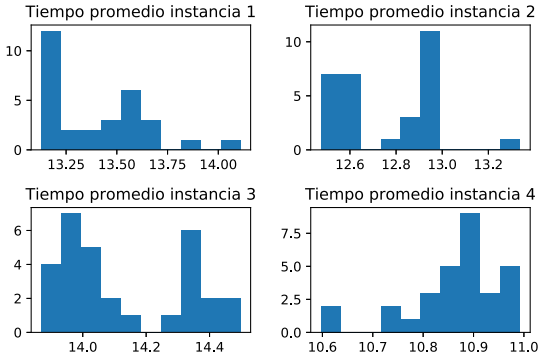


Figura 12: Shortestpath

6. Spanning tree

Las graficas 16,17,18 19 corresponden a las medias con las desviaciones extandar de cada uno de los problemas resueltos con un algoritmo representado por una figura de un color. Los algoritmos para determinar la distancia más corta entre dos nodos así como el máximo flujo son los más sensibles al tiempo cuando aumenta la cantidad de nodos

y vértices del grafo. En ambos casos vértices vs tiempo y nodos vs tiempo se realizó un acercamiento en la zona donde se concentran la mayoría de los tiempos medios entre $(9.725; 9.9775)$.

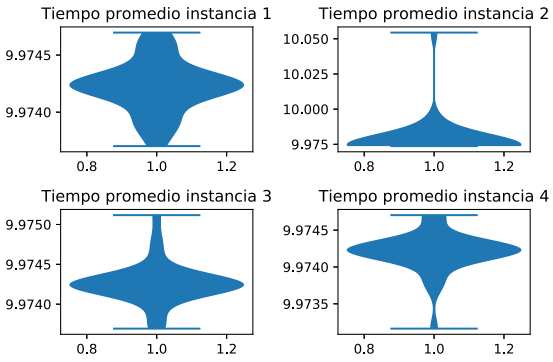


Figura 13: Violinplot spanning tree

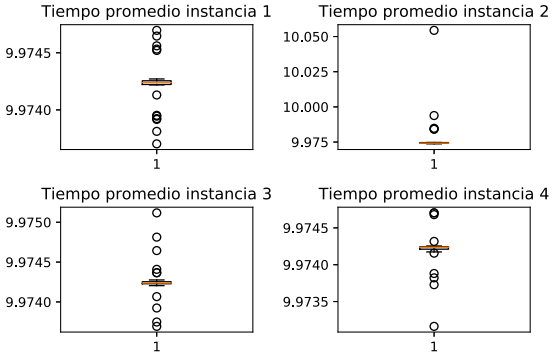


Figura 14: Boxplot spanning tree

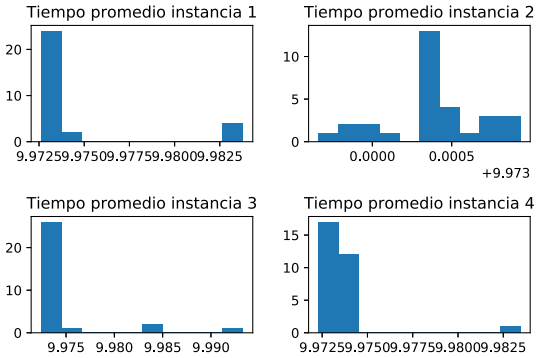


Figura 15: Spanning tree

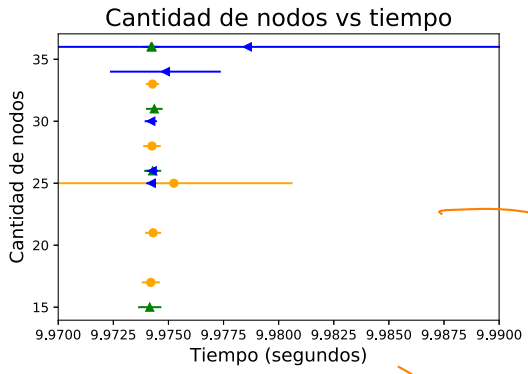


Figura 16: Errors bars nodes

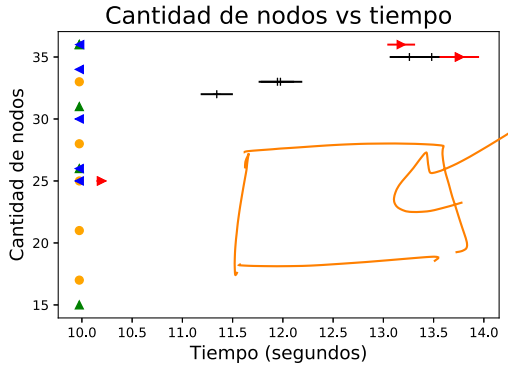


Figura 17: Errors bars nodes

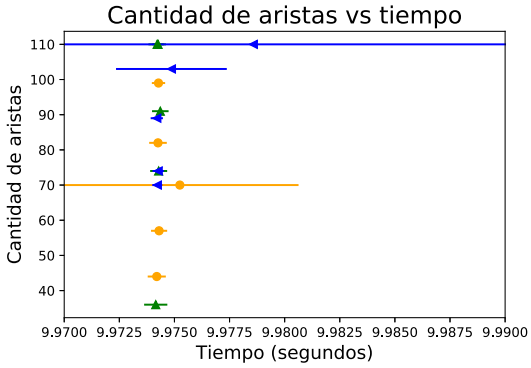


Figura 18: Errors bars edges

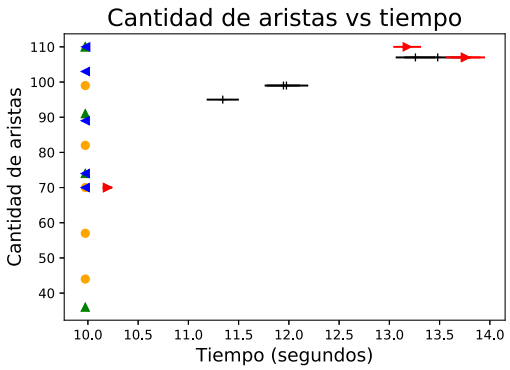


Figura 19: Errors bars edges

Referencias

[1] Janet M Six and Ioannis G Tollis. A framework for circular drawings of networks. In *International Symposium on Graph Drawing*, pages 107–116. Springer, 1999.

[2] Yehuda Koren. On spectral graph drawing. In *International Computing and Combinatorics Conference*, pages 496–508. Springer, 2003.

[3] EW Mayr. Praktikum algorithmen-entwurf (teil 6), nov. 2002, 6–11. *Technische Universität München http://wwwmayr. in. tum. de/lehre/2002WS/algoprak/part6. ps. gz.*

[4] E. Shaeffer. https://elisa.dyndns-web.com/.

Tarea 4

1985269

2 de abril de 2019

1. Descripción del experimento

Se escogieron tres generadores de grafos de la librería NetworkX, con cada un de ellos se generaron cuatro grafos de distinto orden (logarítmico de base 3). De cada orden se obtuvieron diez grafos con densidades distintas. Luego se determinó el flujo máximo de cada grafo con cinco combinaciones de fuente sumidero distintas cinco veces más. Para calcular el flujo máximo se utilizaron tres algoritmos, para cada uno de ellos se realizo el procedimiento anterior.

Se almacenan los tiempos de ejecución de cada uno de los ciclos con el objetivo de estudiar los factores que lo afectan. Otras variables de interés estudiadas son: algoritmo, generador del grafo, cantidad de nodos y densidad.

Los algoritmos utilizados fueron:

- Maximum flow
- Edmonds Karp
- Boykov Kolmogorov

Generadores de Grafos:

- dense gnm random graph
- gnm random graph
- gnp random graph

El objetivo planteado es determinar si las variables de interés influyen en el tiempo de ejecución, para lo cual se realizó un análisis de varianzas y uno de correlación.

A continuación se comparte el código de Python con el que se recopiló la información:

```
Graf=nx.Graph()
rog=0
diccionario_inst_Shortest_path={}
while rog<=4:
    Graf.clear()
    rango=random.randint(random.randint(10,len(matrizadd)),len(matrizadd))
    for i in range(rango):
        for j in range(rango):
            if matrizadd[i,j]!=0:
                matrizadd[j,i]=0
                #Graf.add_weighted_edges_from([(i,j,matrizadd[i,j])])
                Graf.add_edges_from([(i,j)])
    cont=0
    lista_tiempos=[]
    lista_tiempos_completos={}
    tiempo_de_paro=0
    tiempo_inicial=0
    tiempo_final =0
    tiempo_ejecucion=0
    for r in range(30):
        lista_tiempos_completos[r+1]=[]
        tiempo_de_paro=0
        while tiempo_de_paro<1:
            tiempo_inicial = time()
            nx.shortest_path(Graf, source=None, target=None, weight=None,
    method='dijkstra')
            tiempo_final = time()
            tiempo_ejecucion = tiempo_final - tiempo_inicial
            if tiempo_ejecucion>0.0:
                lista_tiempos_completos[r+1].append((tiempo_ejecucion*10000))
            tiempo_de_paro+=tiempo_ejecucion
    guardar_n_e[rog]=[]
    diccionario_inst_Shortest_path[rog]=[]
    for i in lista_tiempos_completos.keys():
         media=np.mean(lista_tiempos_completos[i])
         diccionario_inst_Shortest_path[rog].append(media)
    guardar_n_e['nodos'].append(len(Graf.nodes))
    guardar_n_e['edges'].append(len(Graf.edges))
    guardar_n_e['media'].append(np.mean(diccionario_inst_Shortest_path[rog]))
    guardar_n_e['desv'].append(np.std(diccionario_inst_Shortest_path[rog]))
    rog+=1
```

2. Análisis de los datos

Como es posible observar en los diagrama caja bigote estos tienen una cola, es decir los tiempos se agrupan en un intervalo y luego se obtienen observaciones de tiempos mayores, lo cual puede tener efectos sobre la media ya que está es muy sensible a puntos extremos. Lo que está sucuediendo en términos prácticos es que la mayoría de los tiempos se encuentran en cierto intervalo y cuando va aumentado el orden del grafo, el tiempo de ejecución va aumentando y salta bruscamente. Esto puede ser bien por la variación de las variables independientes combinado por el tiempo que demora el algoritmo de flujo máximo en resolver el problema.

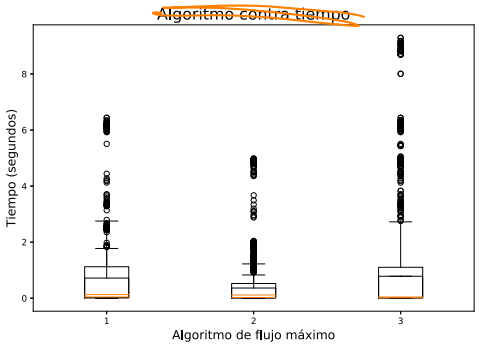


Figura 1: Algoritmo contra Tiempo

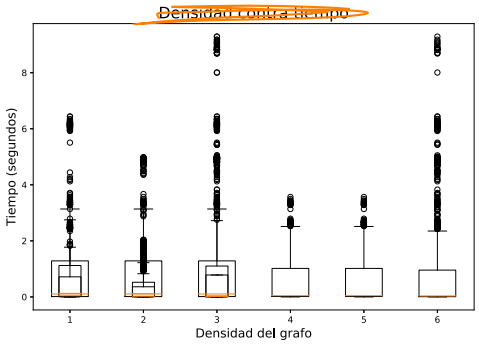


Figura 2: Densidad contra tiempo

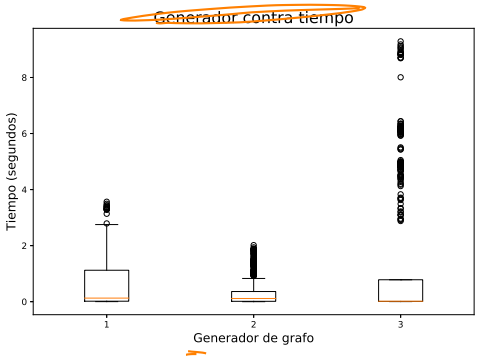


Figura 3: Generador contra grafo

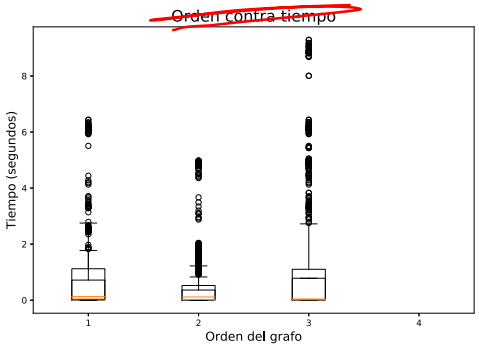


Figura 4: Orden contra tiempo

3. ANOVA

Se realizó un análisis de varianzas para cada uno de las variables con el objetivo de determinar si la medias con respecto al tiempo son diferentes. Como es posible observar en las salidas de la prueba el valor de P es muy pequeño, por lo que se puede afirmar que las medias de las variables con respecto al tiempo son diferentes. Entonces es posible concluir que las variables se relacionan con el tiempo de ejecución.

ANOVA.txt

	sum_sq	df	F	PR>F
Algoritmo	2.994151	1.0	4.623[illegible]5	0.00
Generador	468.400[illegible]6	1.0	723.24[illegible]0	0.00
Orden	2999.13[illegible]7	1.0	4630.90[illegible]8	0.00
Densidad	733.07[illegible]8	1.0	1131.91[illegible]2	0.00
Residual	1162.505304	1795.0	NaN	NaN
c@FancyV[illegible]	1162.505304	1795.0	NaN	NaN

4. Mínimos cuadrados ordinarios

Para estudiar más a fondo la relación entre las variables y el tiempo se realizó una prueba de mínimos cuadrados ordinarios (OLS) por sus siglas en inglés. La salidas se muestran a continuación y el de R-squred indica que con estas variables es posible crear un modelo que elimine el setenta y cinco por ciento de los errores para determinar el tiempo. Del análisis de los P valores se puede reafirmar que la medias de las variables con respecto al tiempo son diferentes. La densidad es la más influyente según este análisis seguida por el generdor y el orden, que aunque es menor se puede afirmar su relación con más confiabilidad que la de la variable algoritmo ya que esta tiene menor P valor. Además el análisis arroja que podría existir una fuerte multicolinealidad.

OLS.txt

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 Tiempo   R-squared:                       0.757
Model:                            OLS   Adj. R-squared:                  0.757
Method:                 Least Squares   F-statistic:                     1399.
Date:                Mon, 01 Apr 2019   Prob F-statistic:               0.00
Time:                        15:58:34   Log-Likelihood:                -2160.6
No. Observations:                1800   AIC:                             4331.
Df Residuals:                    1795   BIC:                             4359.
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -3.0898      0.091    -34.136      0.000      -3.267      -2.912
Algoritmo      0.0500      0.023      2.150      0.032       0.004       0.096
Generador      0.6755      0.025     26.893      0.000       0.626       0.725
Orden          0.0047   6.89e-05     68.051      0.000       0.005       0.005
Densidad       1.9568      0.058     33.644      0.000       1.843       2.071
==============================================================================
Omnibus:                      762.497   Durbin-Watson:                   0.441
ProbOmnibus:                    0.000   Jarque-Bera JB:               6061.181
Skew:                           1.791   ProbJB:                          0.00
Kurtosis:                      11.245   Cond. No.                     2.06e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.06e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

5. Correlación y Comportamiento

En la figura 5 se muestra la matriz de correlación. Es posible observar que las correlaciones con el tiempo de las variables orden y densidad ratificando la información de la prueba anterior. Por útimo en la gráfica 2 se muestran en colores distintos cada uno de los algoritmos estudiados y puede observarse como a medida que aumenta el orden el tiempo crece abruptamente. Cada uno de los algoritmos fue representado con distintas formas.

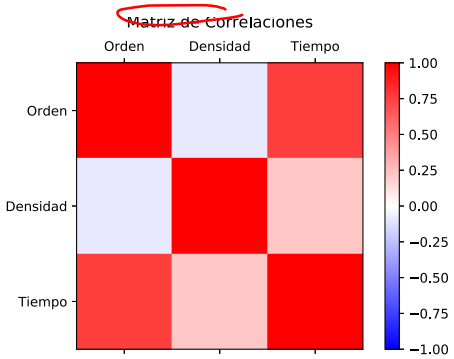


Figura 5: Correlación

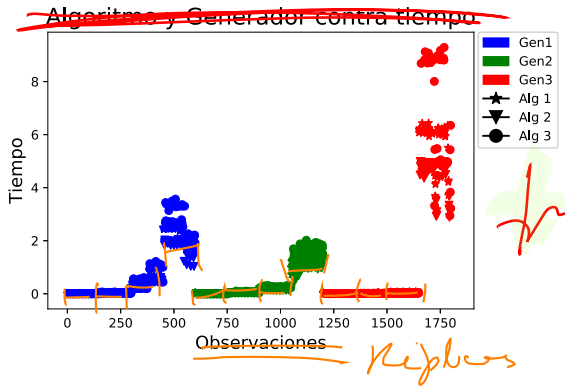


Figura 6: Comportamiento con respecto al tiempo

Referencias

[1] Yehuda Koren. On spectral graph drawing. In *International Computing and Combinatorics Conference*, pages 496–508. Springer, 2003.

[2] E. Schaeffer. https://elisa.dyndns-web.com/.

[3] Python. https://pythonfordatascience.org/anova-python/.

Tarea 5

1985269

29 de abril de 2019

1. Descripción del experimento

Fueron generados cinco grafos de órdenes distintos, luego fueron escogidos dos nodos entre los cuales se calculó el máximo flujo que puede transportarse de uno a otro. Este procedimiento fue repetido con varios pares de nodos y en cada una de las repeticiones se determinaron los coeficientes de argupamiento (clustering), centralidad (closeness centrality, load, centrality), exentricidad (eccentricity) etc. Los pesos de las aristas fueron asignados siguiendo una ditribución normal. El objetivo de este experimento es determinar si existe alguna relación entre dichos coeficientes y el valor de la función objetivo o con el tiempo de ejecución. Así como determinar cuales son los mejores nodos fuentes y sumideros.

2. Gederador de grafos

Fue utilizado el generador Watts–Strogatz small-world, este genera conexiones con una probabilidad predefinida. Se encuentra cierta similitud entre la manera en la que se genera este grafo con asentamientos poblacionales donde alrededor de las ciudades con una probabilidad mayor se fundan nuevos asentamientos y el flujo podría verse como el comercio.

A continuación se comparte el código de Python con el que se recopiló la información:

```
for i in range(5):
    rango=random.randint(ordenes[i],ordenes[i]*2)
    #G=nx.dense_gnm_random_graph(ordenes[i],rango)
    G=nx.watts_strogatz_graph(ordenes[i], int(ordenes[i]/2) , 0.33 , seed=
    None)
    lista=[]
    lista[:]=G.edges
    width=np.arange(len(lista)*1,dtype=float).reshape(len(lista),1)
    for r in range(len(lista)):
        R=np.random.normal(loc=20, scale=5.0, size=None)
        width[r]=R
        G.add_edge(lista[r][0],lista[r][1],capacity=R)
    for w in range(ordenes[i]):
        initial=final=0
        while initial==final:
            initial=random.randint(0,round(len(G.nodes)/2))
            final=random.randint(initial,len(G.nodes)-2)


        tiempo_inicial=time()
        T=nx.maximum_flow(G, initial , final)
        tiempo_final =time()
        tiempo_ejecucion=tiempo_final- tiempo_inicial

        data[contador,2]=nx.clustering(G, nodes=initial)
        data[contador,3]=nx.load_centrality(G, v=initial)
        data[contador,4]=nx.closeness_centrality(G, u=initial)
        data[contador,5]=nx.eccentricity(G, v=initial)
        data[contador,6]=nx.pagerank(G, alpha=0.9)[initial]
#   ------------------------------------------------------------------------------


        data[contador,7]=nx.clustering(G, nodes=final)
        data[contador,8]=nx.load_centrality(G, v=final)
        data[contador,9]=nx.closeness_centrality(G, u=final)
        data[contador,10]=nx.eccentricity(G, v=final)
        data[contador,11]=nx.pagerank(G, alpha=0.9)[final]
#   ------------------------------------------------------------------------------


        data[contador,0]=T[0]
        data[contador,1]=tiempo_ejecucion
        data[contador,12]=ordenes[i]
        contador+=1
```

3. Visualización de los grafos Grafo

A continuación se presentan los grafos generados, los nodos marcados con formas y colores representan la fuente y el sumidero con verde y rojo respentivamente. Las aristas azules representan el flujo y su anchora es proporcional a su capacidad. Es posible observar como los grafo tienen ordenes distintos.

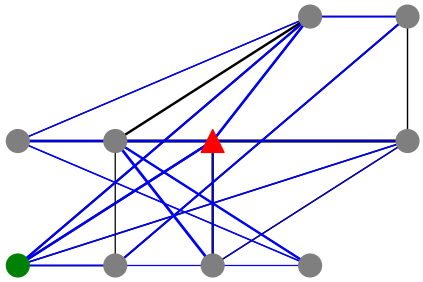


Figura 1: Grafo no dirigido cíclico de orden 10

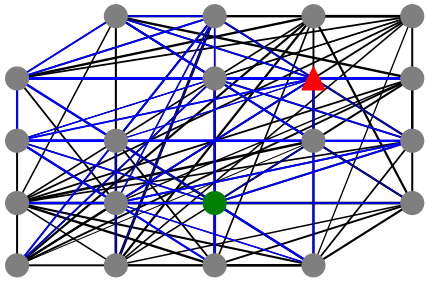


Figura 2: Grafo no dirigido cíclico de orden 20

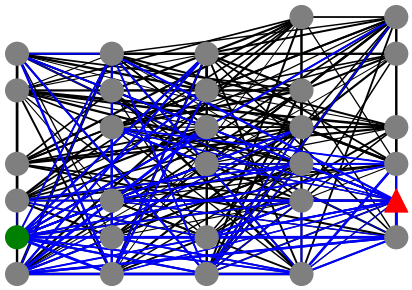


Figura 3: Grafo no dirigido cíclico de orden 30

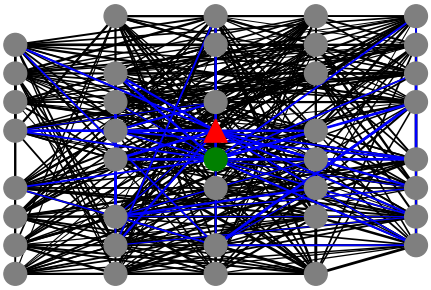


Figura 4: Grafo no dirigido cíclico de orden 40

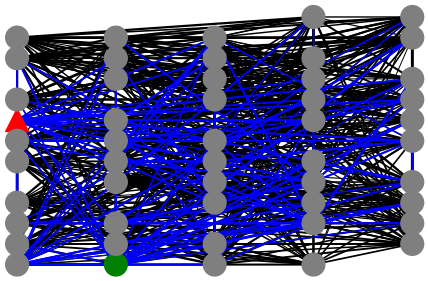
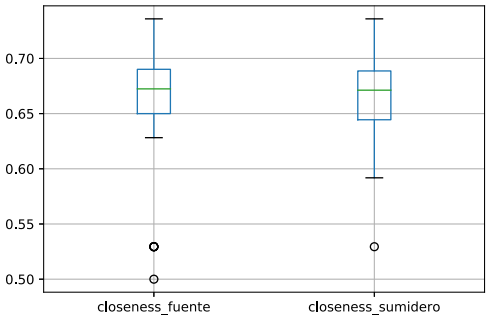


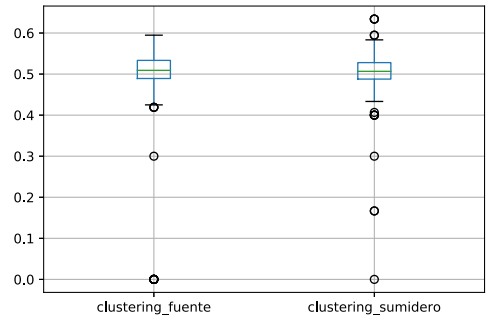
Figura 5: Grafo no dirigido cíclico de orden 50

4. Análisis de los datos

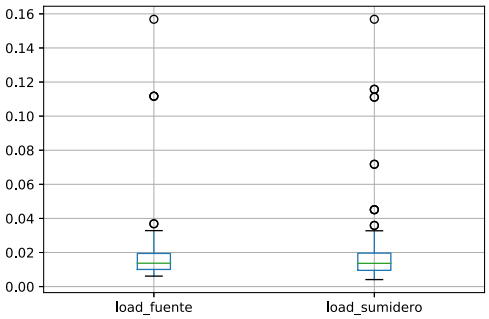
Los gráficos de caja y bigotes muestran como los datos obtenidos para cada uno de los nodos presentan ciertas similitudes, esto producto a que dependen del generador utilizado, es decir si fuera realizado un análisis entre distintos generadores resultaría una relación entre ellos y las caractirístias medidas para los nodos. El tiempo de ejecución registrado para el algoritmo de flujo máximo es pequeño para el tamaño de las instancias estudiadas. En la matriz de correlaciones el tiempo fila, columna 1 no se encuentra correlacionado fuertemente con ningúna otra variable, esto será ratificado más adelante. Aun así sí existen correlaciones fuertes entre las otras variables.

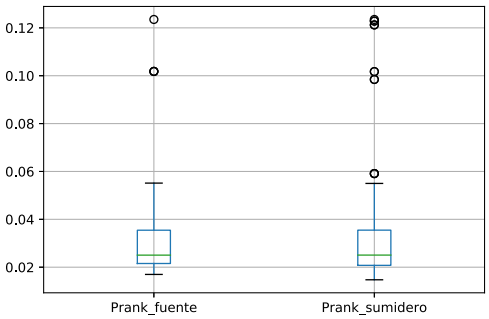

0.70
0.65
0.60
0.55
0.50
closeness_fuente
closeness_sumidero

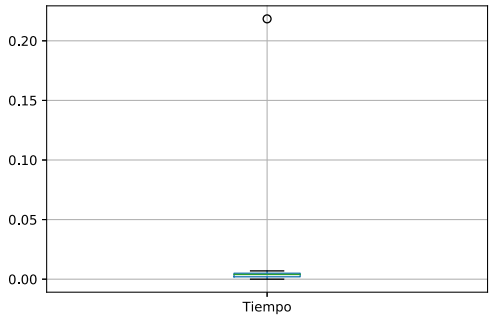
(a)


0.6
0.5
0.4
0.3
0.2
0.1
0.0
clustering_fuente
clustering_sumidero

(b)


0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
load_fuente
load_sumidero


0.12
0.10
0.08
0.06
0.04
0.02
Prank_fuente
Prank_sumidero



5. Análisis de varianza

Se realizó una prueba ANOVA para demostrar desigualdad entre las medias. Como indican los resultados de la prueba los valores p son pequeños las medias no son iguales, por lo cual se procedió a utilizar una prueba de mínimos cuadrados ordinarios a ver si con las variables resgistradas se pueden explicar los valores de la función objetivo.

ANOVA.txt

	sum_sq	...	PR>F
clustering_fuente	14340	...	0
load_fuente	60054	...	0
closeness_fuente	101311	...	0
eccentricity_fuente	140950	...	0
Prank_fuente	132493	...	0
clustering_sumidero	10	...	0
load_sumidero	15165	...	0
closeness_sumidero	505	...	0
eccentricity_sumidero	151097	...	0
Prank_sumidero	146	...	0.07
Residual	272297	...	NaN
c@FancyVerbLinee	272297	...	NaN

6. Mínimos cuadrados ordinarios

Los resultados de la regresión indican que con las variables medidas es posible explicar el comportamiento de los valores del flujo máximo con un error aceptable, el modelo hace una buena predicción. Las variables clustering no son significativas en el modelo al igual que load y page rank sumidero. Las caractirísticas más influyentes de los nodos fuentes son load, closeness y eccentricity mientras que los sumideros closeness y load en ese orden.

En el segundo experimento se realizó la misma prueba para el tiempo de ejecución y el resultado fue como se esperaba que el modelo no explica el comportamiento del tiempo ya que este depende de otras variables. Como se puede observar en la matriz de correlación el tiempo se correlaciona con el valor objetivo y esto tiene sentido ya que si existen varias aristas por las cuales pasa flujo la exploración es más grande y el tiempo de ejecución aumenta.

OLS.txt

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                     FO   R-squared:                       0.920
Model:                            OLS   Adj. R-squared:                  0.915
Method:                 Least Squares   F-statistic:                     160.5
Date:                Sat, 20 Apr 2019   Prob F-statistic:           3.61e-71
Time:                        12:27:00   Log-Likelihood:                -739.54
No. Observations:                 150   AIC:                             1501.
Df Residuals:                     139   BIC:                             1534.
Df Model:                          10
Covariance Type:            nonrobust
============================================================================================
                             coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
Intercept             -1268.0017    223.711     -5.668      0.000   -1710.318    -825.686
clustering_fuente        -7.3702     76.549     -0.096      0.923    -158.722     143.982
load_fuente            6235.1056    565.337     11.029      0.000    5117.334    7352.877
closeness_fuente        718.4359    163.935      4.382      0.000     394.308    1042.564
eccentricity_fuente     222.4259     57.859      3.844      0.000     108.029     336.823
Prank_fuente          -1.034e+04   1317.746     -7.848      0.000   -1.29e+04   -7736.633
clustering_sumidero      39.8599     79.430      0.502      0.617    -117.187     196.907
load_sumidero           200.1120    467.998      0.428      0.670    -725.203    1125.427
closeness_sumidero     1156.9662    176.495      6.555      0.000     808.005    1505.928
eccentricity_sumidero    78.2676     27.680      2.828      0.005      23.540     132.996
Prank_sumidero        -1989.1513   1327.265     -1.499      0.136   -4613.390     635.087
==============================================================================
Omnibus:                        1.709   Durbin-Watson:                   1.384
ProbOmnibus:                    0.425   Jarque-Bera JB:                  1.486
Skew:                          -0.243   ProbJB:                          0.476
Kurtosis:                       3.038   Cond. No.                     2.18e+03
==============================================================================
c@FancyVerbLinee==============================================================
```

Olsm.txt

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 Tiempo   R-squared:                       0.125
Model:                            OLS   Adj. R-squared:                  0.062
Method:                 Least Squares   F-statistic:                     1.978
Date:                Sat, 20 Apr 2019   Prob F-statistic:             0.0400
Time:                        12:27:00   Log-Likelihood:                485.89
No. Observations:                 150   AIC:                           -949.8
Df Residuals:                     139   BIC:                           -916.7
Df Model:                          10
Covariance Type:            nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
Intercept                        0.0893      0.063      1.409      0.161      -0.036       0.215
clustering_fuente               -0.0162      0.022     -0.747      0.456      -0.059       0.027
load_fuente                      0.0033      0.160      0.021      0.984      -0.313       0.320
closeness_fuente                 0.0226      0.046      0.486      0.628      -0.069       0.114
eccentricity_fuente             -0.0082      0.016     -0.499      0.618      -0.041       0.024
Prank_fuente                    -0.1936      0.373     -0.519      0.605      -0.931       0.544
clustering_sumidero              0.0083      0.022      0.367      0.714      -0.036       0.053
load_sumidero                    0.0069      0.133      0.052      0.959      -0.255       0.269
closeness_sumidero              -0.0917      0.050     -1.835      0.069      -0.190       0.007
eccentricity_sumidero           -0.0064      0.008     -0.820      0.413      -0.022       0.009
Prank_sumidero                   0.0528      0.376      0.141      0.888      -0.690       0.796
==============================================================================
Omnibus:                      143.935   Durbin-Watson:                   2.152
ProbOmnibus:                    0.000   Jarque-Bera JB:               1529.659
Skew:                           3.756   ProbJB:                           0.00
Kurtosis:                      16.722   Cond. No.                     2.18e+03
==============================================================================
c@FancyVerbLinee==============================================================
```

7. Conclusiones

Fue realizada la experimentción propuesta y los resultados fueron analizados, dicho análisis arrojó que con las variables medidas es posible encontrar un modelo que explique el valor objetivo del algoritmo del flujo máximo utilizado. Los mejores nodos fuentes son aquellos que tienen un mayor valor de load, closeness y eccentricity. Mientras que los mejores sumidero son aquellos que presentan mayor valor de closeness. Además se comprobó que no existe una correlación fuerte entre el tiempo de ejecución y estas variables aunque existe cierta relación positiva entre el tiempo y el valor del flujo máximo.

Referencias

[1] Janet M Six and Ioannis G Tollis. A framework for circular drawings of networks. In *International Symposium on Graph Drawing*, pages 107–116. Springer, 1999.

[2] Yehuda Koren. On spectral graph drawing. In *International Computing and Combinatorics Conference*, pages 496–508. Springer, 2003.

[3] EW Mayr. Praktikum algorithmen-entwurf (teil 6), nov. 2002, 6–11. *Technische Universität München http://wwwmayr. in. tum. de/lehre/2002WS/algoprak/part6. ps. gz.*

[4] E. Shaeffer. https://elisa.dyndns-web.com/.

Tarea 5

1985269

3 de junio de 2019

1. Descripción del experimento

Fueron generados cinco grafos , luego fueron escogidos dos nodos entre los cuales se calculó el máximo flujo que puede transportarse de uno a otro. Este procedimiento fue repetido con varios pares de nodos y en cada una de las repeticiones se determinaron los coeficientes de argupamiento (clustering), centralidad (closeness centrality, load, centrality), excentricidad (eccentricity) etc. Los pesos de las aristas fueron asignados siguiendo una distribución normal. El objetivo de este experimento es determinar si existe alguna relación entre dichos coeficientes y el valor de la función objetivo o con el tiempo de ejecución. Así como determinar cuáles son los mejores nodos fuentes y sumideros.

2. Gederador de grafos

Fue utilizado el generador Watts–Strogatz small-world, este genera conexiones con una probabilidad predefinida. Se encuentra cierta similitud entre la manera en la que se genera este grafo con asentamientos poblacionales donde alrededor de las ciudades con una probabilidad mayor se fundan nuevos asentamientos y el flujo podría verse como el comercio.

A continuación se comparte el código de Python con el que se recopiló la información:

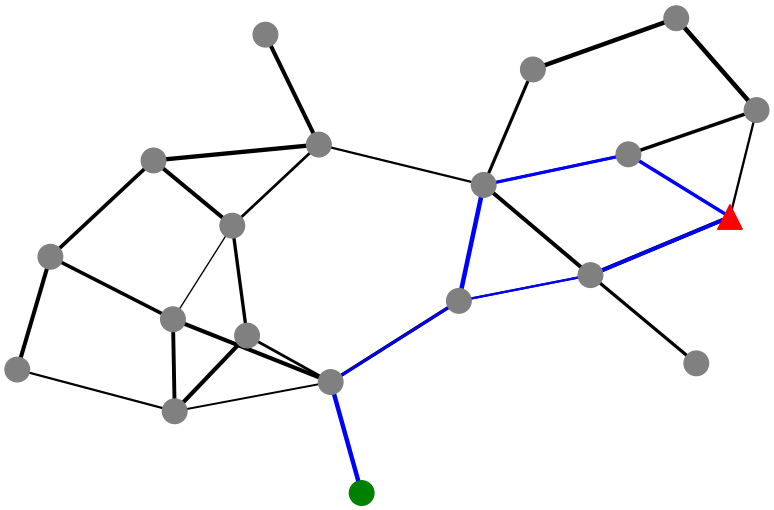
```
for i in range(5):
    rango=random.randint(ordenes[i],ordenes[i]*2)
    #G=nx.dense_gnm_random_graph(ordenes[i],rango)
    G=nx.watts_strogatz_graph(ordenes[i], int(ordenes[i]/2) , 0.33 , seed=None)
    lista=[]
    lista[:]=G.edges
    width=np.arange(len(lista)*1,dtype=float).reshape(len(lista),1)
    for r in range(len(lista)):
        R=np.random.normal(loc=20, scale=5.0, size=None)
        width[r]=R
        G.add_edge(lista[r][0],lista[r][1],capacity=R)
    for w in range(ordenes[i]):
        initial=final=0
        while initial==final:
            initial=random.randint(0,round(len(G.nodes)/2))
            final=random.randint(initial,len(G.nodes)-2)


        tiempo_inicial=time()
        T=nx.maximum_flow(G, initial , final)
        tiempo_final =time()
        tiempo_ejecucion=tiempo_final- tiempo_inicial

        data[contador,2]=nx.clustering(G, nodes=initial)
        data[contador,3]=nx.load_centrality(G, v=initial)
        data[contador,4]=nx.closeness_centrality(G, u=initial)
        data[contador,5]=nx.eccentricity(G, v=initial)
        data[contador,6]=nx.pagerank(G, alpha=0.9)[initial]
#____________________________________________________________________________
        data[contador,7]=nx.clustering(G, nodes=final)
        data[contador,8]=nx.load_centrality(G, v=final)
        data[contador,9]=nx.closeness_centrality(G, u=final)
        data[contador,10]=nx.eccentricity(G, v=final)
        data[contador,11]=nx.pagerank(G, alpha=0.9)[final]
#____________________________________________________________________________
        data[contador,0]=T[0]
        data[contador,1]=tiempo_ejecucion
        data[contador,12]=ordenes[i]
        contador+=1
```

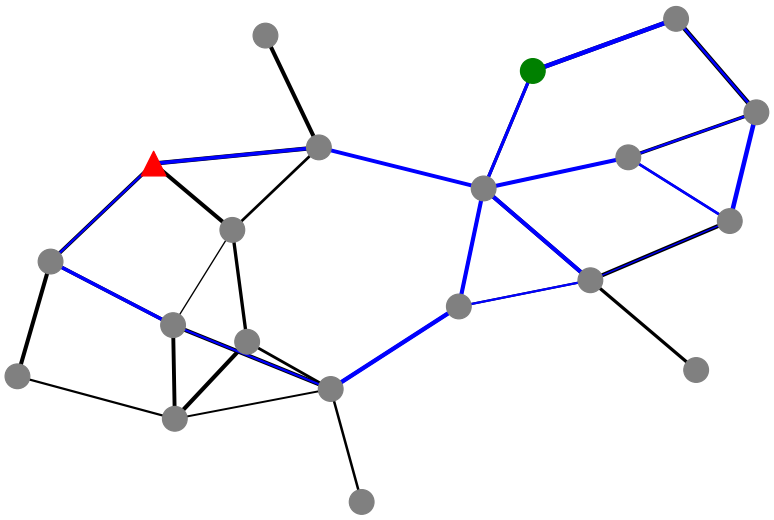
3. Visualización de los grafos Grafo

A continuación se presentan los grafos generados, los nodos marcados con formas y colores representan la fuente y el sumidero con verde y rojo respectivamente. Las aristas azules representan el flujo y su anchura es proporcional a su capacidad.



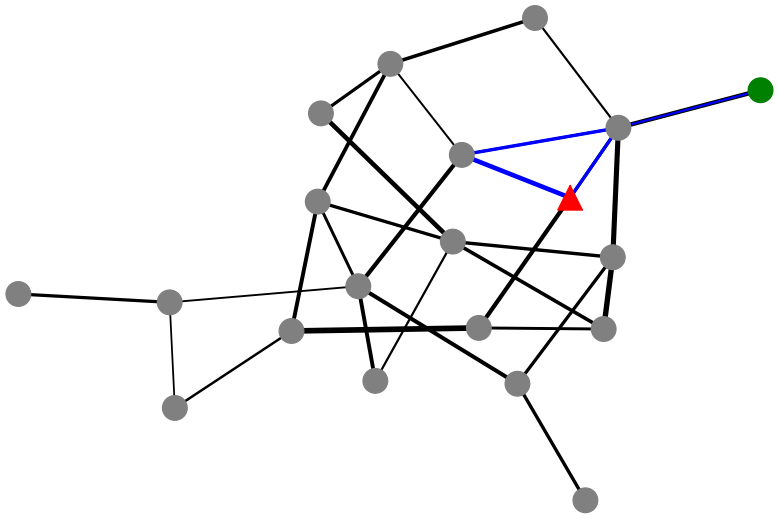
(a) Figura 1

Figura 1: Grafo 1 Mínimo valor de flujo entre dos nodos



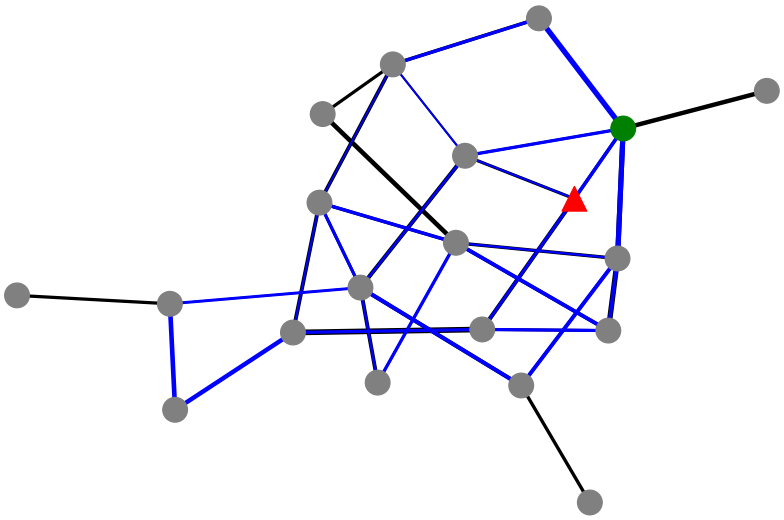
(a) Figura 2

Figura 2: Grafo 1 Máximo valor de flujo entre dos nodos



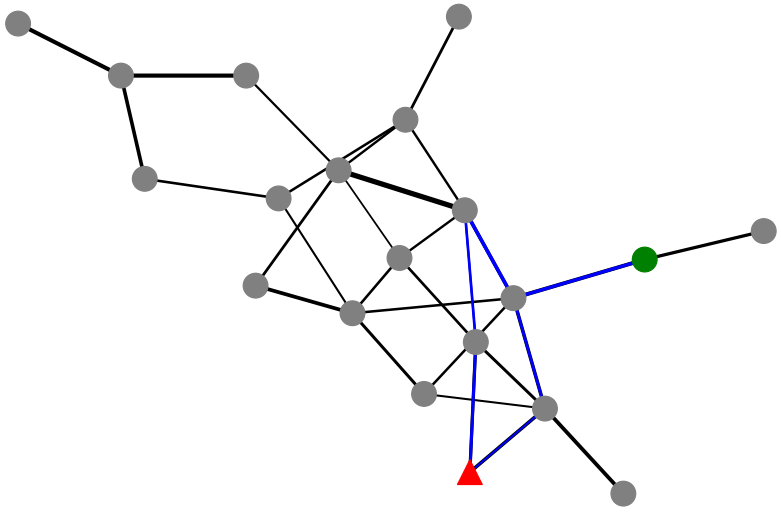
(a) Figura 1

Figura 3: Grafo 2 Mínimo valor de flujo entre dos nodos



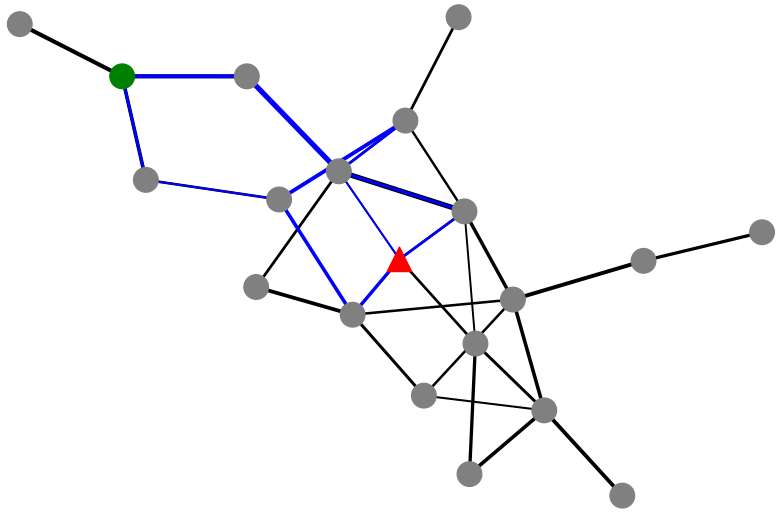
(a) Figura 2

Figura 4: Grafo 2 Máximo valor de flujo entre dos nodos



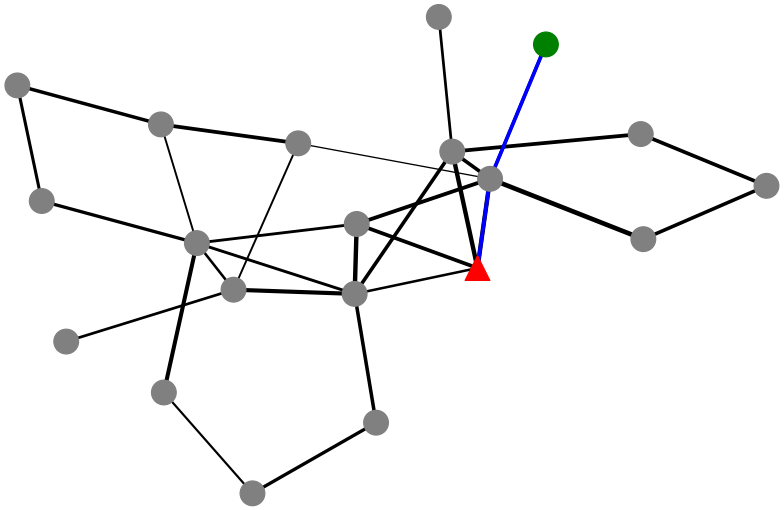
(a) Figura 1

Figura 5: Grafo 3 Mínimo valor de flujo entre dos nodos



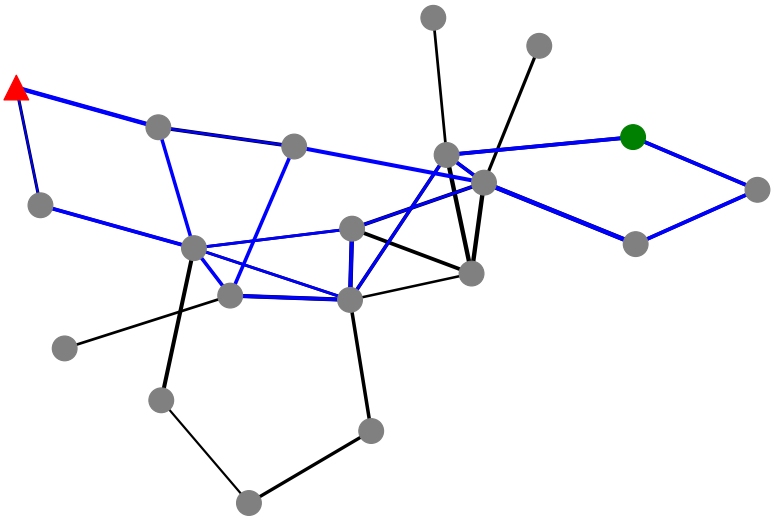
(a) Figura 2

Figura 6: Grafo 3 Máximo valor de flujo entre dos nodos



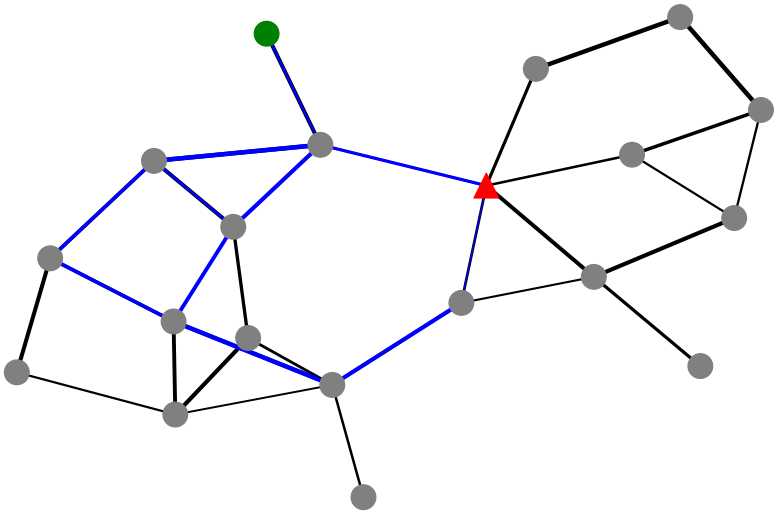
(a) Figura 1

Figura 7: Grafo 4 Mínimo valor de flujo entre dos nodos



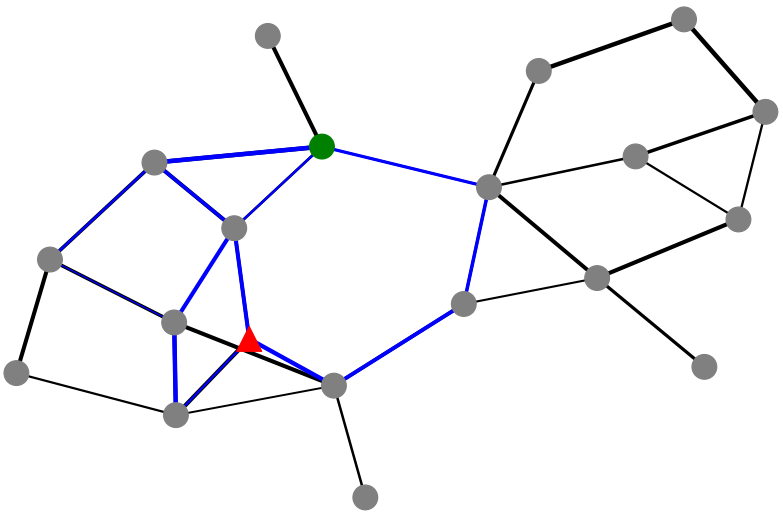
(a) Figura 2

Figura 8: Grafo 4 Máximo valor de flujo entre dos nodos



(a) Figura 1

Figura 9: Grafo 5 Mínimo valor de flujo entre dos nodos

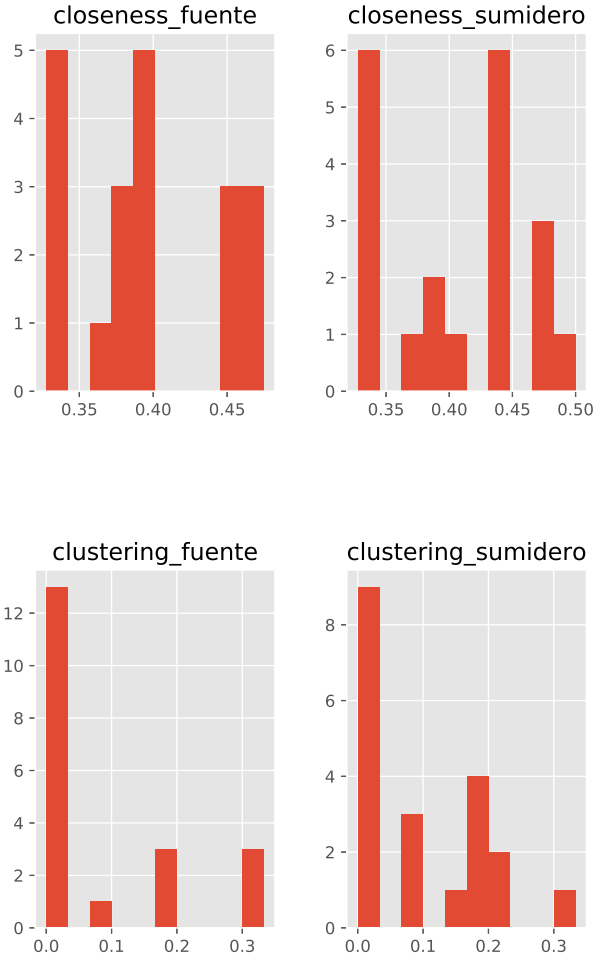


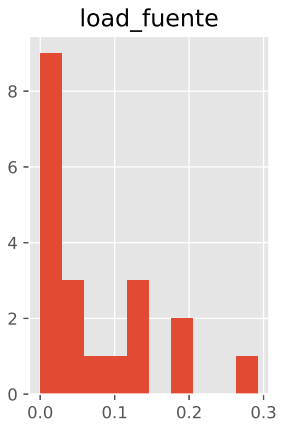
(a) Figura 2

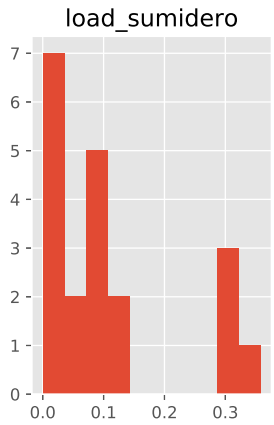
Figura 10: Grafo 5 Máximo valor de flujo entre dos nodos

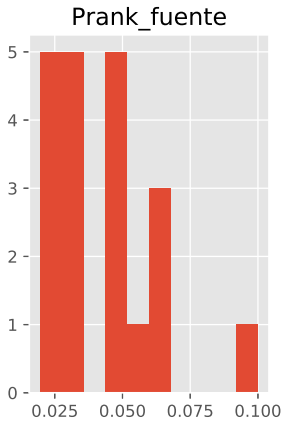
4. Análisis de los datos

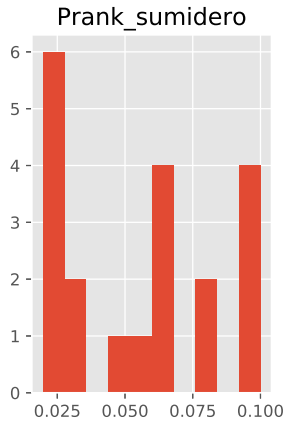
En los histogramas presentados se muestran las características de los nodos fuentes y sumideros, estas no son similares en este caso y no se aproximan a distribución alguna. El tiempo de ejecución registrado para el algoritmo de flujo máximo es pequeño para el tamaño de las instancias estudiadas.

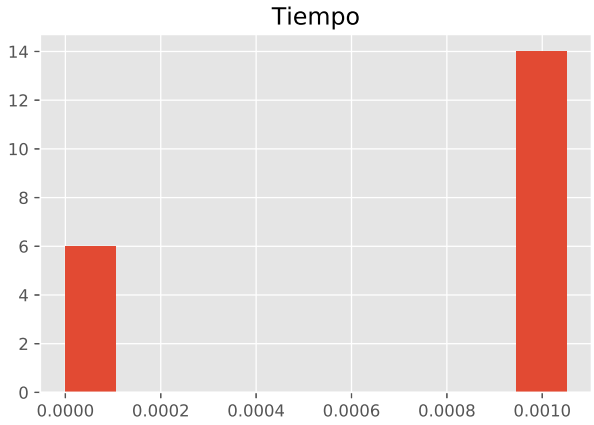



load_fuente
0
2
4
6
8
0.0
0.1
0.2
0.3


load_sumidero
0
1
2
3
4
5
6
7
0.0
0.1
0.2
0.3


Prank_fuente
0
1
2
3
4
5
0.025
0.050
0.075
0.100


Prank_sumidero
0
1
2
3
4
5
6
0.025
0.050
0.075
0.100


Tiempo
0
2
4
6
8
10
12
14
0.0000
0.0002
0.0004
0.0006
0.0008
0.0010

5. Análisis de varianza

Se realizó una prueba ANOVA para demostrar desigualdad entre las medias. Como indican los resultados de la prueba los valores p de algunas características son pequeños, es decir existen diferencias entre las medias, se realizó un prueba de mínimos cuadrados ordinarios para ver si con las variables registradas se pueden explicar los valores de la función objetivo.

ANOVA.txt

	sum_sq	df	F	PR>F
clustering_fuente	1.810887	1.0	0.019467	0.892107
load_fuente	183.463600	1.0	1.972273	0.193764
closeness_fuente	515.844978	1.0	5.545443	0.042960
eccentricity_fuente	42.491256	1.0	0.456790	0.516117
Prank_fuente	181.616210	1.0	1.952413	0.195816
clustering_sumidero	38.706353	1.0	0.416102	0.534984
load_sumidero	6.551205	1.0	0.070427	0.796695
closeness_sumidero	10.011648	1.0	0.107627	0.750365
eccentricity_sumidero	18.773343	1.0	0.201817	0.663881
Prank_sumidero	0.638830	1.0	0.006868	0.935768
Residual	837.192698	9.0	NaN	NaN
c@FancyVerbLinee	837.192698	9.0	NaN	NaN

6. Mínimos cuadrados ordinarios

Los resultados de la regresión indican que con las variables medidas es posible explicar el comportamiento de los valores del flujo máximo con un error aceptable, el modelo hace una buena predicción. Las variables: clossenes centrality, eccentricity y page rank de la fuente no son significativas en el modelo al igual que load. Las caractirísticas más influyentes de los nodos fuentes son el clustering y clossenes centrality, mientras que en el sumidero todas las variables presentan un aporte significativo.

En el segundo experimento se realizó la misma prueba para el tiempo de ejecución. Resultando que la variable eccentricity está relacionada positivamente con el tiempo de

ejecución y el modelo compuesto por estas variables explica aceptablemente el tiempo de ejecución.

OLS.txt

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                     FO   R-squared:                       0.901
Model:                            OLS   Adj. R-squared:                  0.791
Method:                 Least Squares   F-statistic:                     8.177
Date:                Mon, 03 Jun 2019   Prob F-statistic:             0.00206
Time:                        17:12:02   Log-Likelihood:                -57.624
No. Observations:                  20   AIC:                             137.2
Df Residuals:                       9   BIC:                             148.2
Df Model:                          10
Covariance Type:            nonrobust
================================================================================================
                                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
Intercept                -110.3618    150.000     -0.736      0.481    -449.685     228.962
clustering_fuente         -29.1025     22.766     -1.278      0.233     -80.602      22.397
load_fuente              -172.1220     48.577     -3.543      0.006    -282.011     -62.233
closeness_fuente          167.3689    254.120      0.659      0.527    -407.491     742.229
eccentricity_fuente        -1.4779     14.333     -0.103      0.920     -33.901      30.945
Prank_fuente              424.1877    827.367      0.513      0.621   -1447.448    2295.823
clustering_sumidero       -37.0497     29.550     -1.254      0.242    -103.897      29.798
load_sumidero            -140.6549     39.343     -3.575      0.006    -229.656     -51.654
closeness_sumidero        110.1839     69.271      1.591      0.146     -46.519     266.887
eccentricity_sumidero       3.1426      2.853      1.102      0.299      -3.310       9.596
Prank_sumidero           1000.1904    305.329      3.276      0.010     309.488    1690.893
==============================================================================
Omnibus:                        4.710   Durbin-Watson:                   1.599
Prob(Omnibus):                  0.095   Jarque-Bera (JB):                2.613
Skew:                          -0.807   Prob(JB):                        0.271
Kurtosis:                       3.726   Cond. No.                     4.53e+03
==============================================================================
```

Olsm.txt

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 Tiempo   R-squared:                       0.718
Model:                            OLS   Adj. R-squared:                  0.404
Method:                 Least Squares   F-statistic:                     2.288
Date:                Mon, 03 Jun 2019   Prob F-statistic:              0.114
Time:                        17:25:14   Log-Likelihood:                139.12
No. Observations:                  20   AIC:                            -256.2
Df Residuals:                       9   BIC:                            -245.3
Df Model:                          10
Covariance Type:            nonrobust
================================================================================================
                                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
Intercept                      0.0011      0.008      0.134      0.896      -0.017       0.019
clustering_fuente             -0.0009      0.001     -0.752      0.471      -0.004       0.002
load_fuente                    0.0023      0.003      0.903      0.390      -0.004       0.008
closeness_fuente               0.0056      0.014      0.416      0.687      -0.025       0.036
eccentricity_fuente            0.0002      0.001      0.236      0.819      -0.002       0.002
Prank_fuente                  -0.0183      0.044     -0.415      0.688      -0.118       0.082
clustering_sumidero           -0.0013      0.002     -0.794      0.447      -0.005       0.002
load_sumidero                  0.0012      0.002      0.557      0.591      -0.004       0.006
closeness_sumidero            -0.0023      0.004     -0.616      0.553      -0.011       0.006
eccentricity_sumidero         -0.0003      0.000     -2.062      0.069      -0.001    3.05e-05
Prank_sumidero                -0.0055      0.016     -0.340      0.742      -0.042       0.031
==============================================================================
Omnibus:                        0.467   Durbin-Watson:                   1.967
Prob(Omnibus):                  0.792   Jarque-Bera (JB):                0.003
Skew:                           0.001   Prob(JB):                        0.998
Kurtosis:                       3.064   Cond. No.                     4.53e+03
==============================================================================
```

7. Conclusiones

Fue realizada la experimentación propuesta y los resultados fueron analizados, dicho análisis arrojó que con las variables medidas es posible encontrar un modelo que explique el valor del flujo máximo y el tiempo de ejecución. Los mejores nodos fuentes son aquellos que tienen un mayor valor de load centrality y clustering. Mientras que los mejores sumideros son aquellos que tienen un alto valor de pagerank y load centrality. Resumiendo el tiempo de ejecución y los valores del flujo máximo se encuentran relacionados con las características de los nodos.

Referencias

[1] Janet M Six and Ioannis G Tollis. A framework for circular drawings of networks. In *International Symposium on Graph Drawing*, pages 107–116. Springer, 1999.

[2] Yehuda Koren. On spectral graph drawing. In *International Computing and Combinatorics Conference*, pages 496–508. Springer, 2003.

Tarea 6

1985269

3 de junio de 2019

1. Introducción

El presente trabajo tiene como objetivo a partir de una red decidir cuáles son los mejores nodos hub así como los nodos críticos que están directamente relacionados con el flujo en la red. Un nodo hub logístico es el lugar donde se reúnen las cargas con la finalidad de ser redistribuidas, en pocas palabras es un puerto o aeropuerto que funcionan como centros de conexiones y distribución. Una red de transporte tiene una capacidad de flujo máxima entre cada uno de sus nodos y un costo asociado a dicho flujo. Un buen nodo hub es aquel que distribuye este flujo de forma tal que los costos sean los menores posibles.

2. Descripción del experimento

Para determinar posibles nodos hubs se analizó la variación del flujo máximo, para lo cuál se generó una red de doscientos nodos. Los nodos de menor grado fueron divididos y conectados con una fuente y un sumidero ficticios, a cada arista que se generó se les asignaron capacidad infinita y costo cero. Las demás aristas del grafo tienen un costo y una capacidad generadas aleatoriamente siguiendo una distribución normal.

Fue calculado el máximo flujo de la red desde la fuente hasta el sumidero y luego se fueron eliminando nodos distintos iterativamente para registrar la variación del flujo. De esta forma se determinan cuáles son los nodos mas influyentes en el flujo de la red. Este mismo procedimiento fue realizado con el costo para este se calculó el camino mínimo y se encontraron los nodos que mayor variación del costo provocan al ser eliminados.

Además fue realizada una búsqueda en anchura para determinar cuáles son los nodos mas conectados sumando la cantidad de conexiones hasta el segundo nivel de la búsdqueda. Los tres aspectos tenidas en cuenta fueron: conectividad, variaciones del flujo y de los costos.

3. Variación del flujo

En la figura 1 se muestra la red generada, el desvanecido de los nodos está dado por el mayor de los costos de ir de un nodo hacia los demás y fue calculado con un algoritmo de la librería NetworkX .

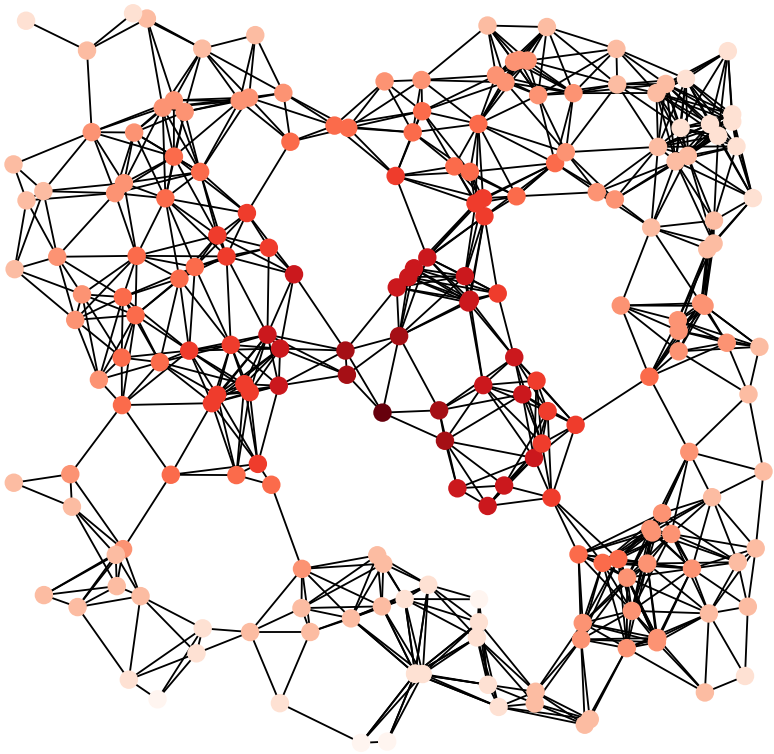


Figura 1: Red de estudio

A continuación se comparte el código de Python del algoritmo utilizado para determinar las variaciones de flujo:

```
T=nx.maximum_flow(G, 'fuente', 'sumidero')
for i in range(rango):
    try:
        H.remove_node(degree[i][1])
        t=nx.maximum_flow(H,'fuente', 'sumidero')
        H=G.copy()
        desv.append((int(T[0]-t[0]),degree[i][1]))
    except:
        H=G.copy()
        desv.append((999999,degree[i][1]))
```

Los resultado obtenidos se muestran en la figura 3 los nodos con mayor intensidad de color son aquellos que al ser removidos del grafo tuvieron una mayor influencia en la

variación del flujo. Los nodos de mayor intensidad se encuentran en las zonas más densas del grafo, estos tienen un mayor número de conexiones y son candidatos claros a ser hubs ya que cumplen con dos de la condiciones definidas previamente.

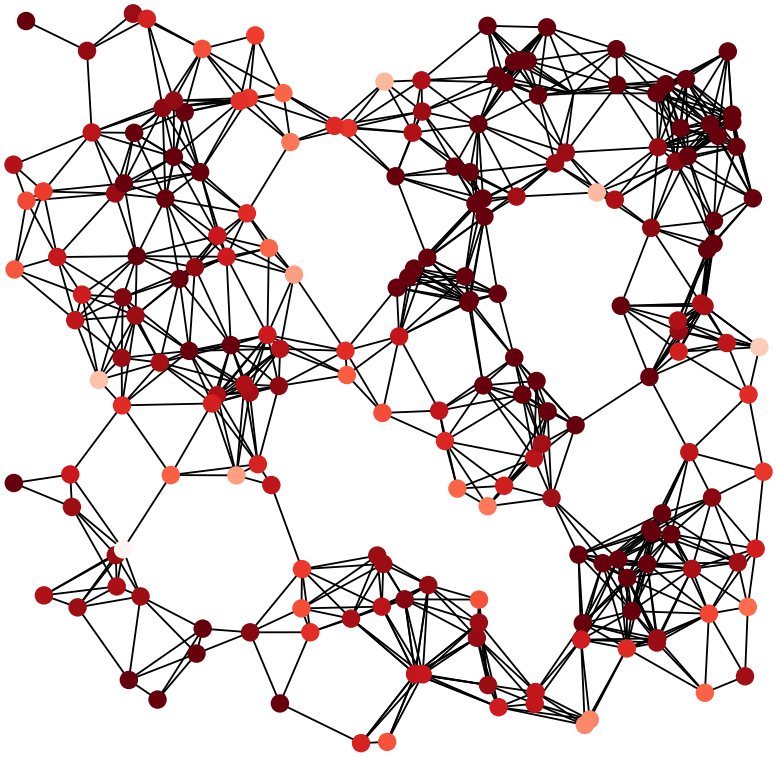


Figura 2: Variación del flujo

4. Variación de los costos

El costo es otro de los indicadores a tener en cuenta. Se realizó un análisis similar al anterior pero esta vez se calculó el camino más corto entre cualesquiera par de nodos y se obtuvieron los resultados que muestra la figura 4. Nuevamente los que mayor variación provocan en la red, al ser removidos, son los nodos de mayor intensidad de color. En este caso tenemos menor concentración de nodos en las zonas densas del grafo. Aparecen nodos en la perifería que tienen un menor grado estos resultan ser nodos críticos ya que de eliminarlos los caminos mínimos tienen una gran variación. Esto debido a que son los enlaces entre las zonas más densas de la red.

```
T=nx.shortest_path(G, 'fuente', 'sumidero', weight=True, method='dijkstra')
for i in range(len(T)-1):
    Short+=H[T[i]][T[i+1]]['weight']
for i in Grafo.nodes:
     try:
         H.remove_node(degree[i][1])
         t=nx.shortest_path(H, 'fuente', 'sumidero', weight=True, method='
    dijkstra')
         for j in range(len(t)-1):
             short+=H[t[j]][t[j+1]]['weight']
         H=G.copy()
         desv1.append((int(short-Short),degree[i][1]))
     except:
         H=G.copy()
         desv1.append((9999999,degree[i][1]))
```

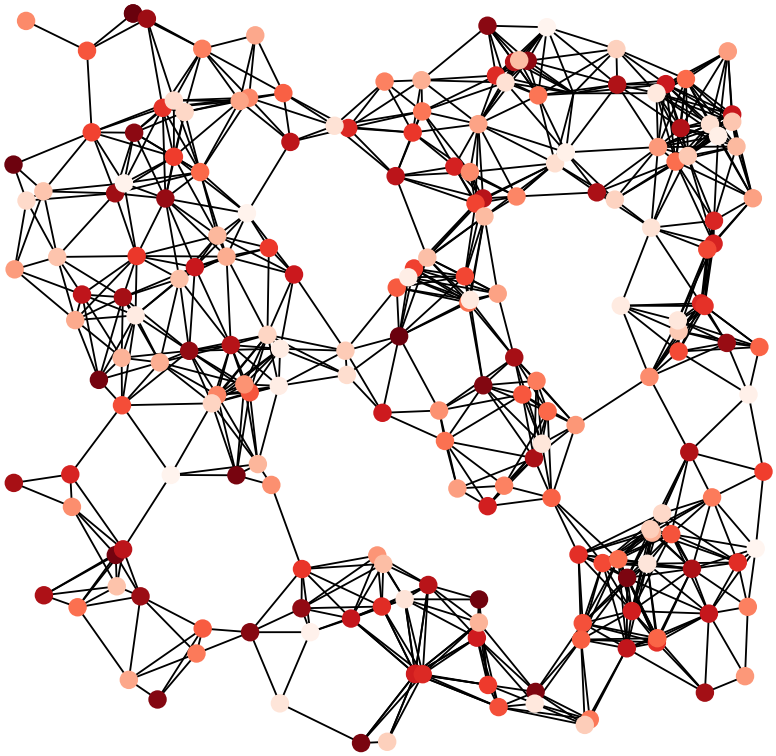


Figura 3: Variación de los costos

5. Conectividad de los nodos

Por último evaluaremos la condición de conectividad para lo cual se realizó una búsqueda en profundidad y se encontraron la cantidad de conexiones de cada nodo hasta el segundo nivel de la búsqueda. Nuevamente como era de esperar los nodos de mayor conectividad se encuentra en las zonas más desnsas del grafo 5.

```
conect=[]
for i in Grafo.nodes:
    K=nx.bfs_tree(Grafo,i)
    conectividad=0
    for j in list(K[i]):
        conectividad+=len(list(K[j]))
    conect.append((conectividad+len(K[i]),degree[i][1]))
```

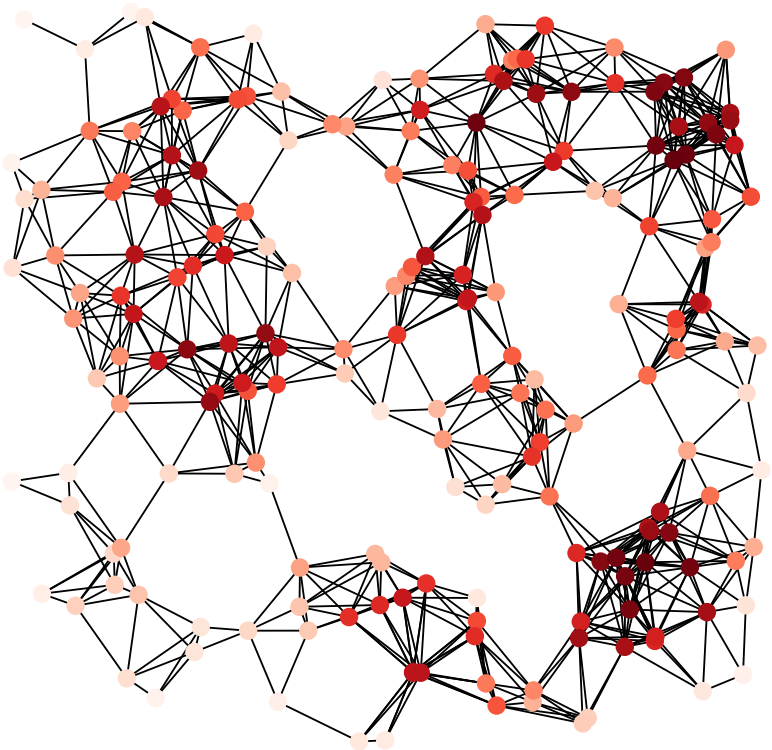


Figura 4: Nodos de mayor conectividad

6. Determinación de los posible hubs en la red

Ya teniendo los tres indicadores que describen el comportamiento de los nodos en cuanto a variación del costo, variación del flujo máximo y conectividad, se procedió a encontrar los nodos que mejores características tuvieran en los tres conjuntos. Cada una de las variaciones se calculó de forma tal que los valores por nodo están relacionados de manera directa con su importancia en la red, es decir a mayor valor de desviación el nodo en cuestión es mejor candidato para hub. Para encontrar los nodos con mejores caraterísticas fueron promediados los indicadores por nodos en la figura 6 se muestra el resultado. Así es que al menos en cada zona densa del grafo hay un nodo candidato a hub, hay otro conjunto de nodos de menor grado que enlazan dichas zonas que de ser eliminados el comportamiento de la red cambia en cuanto al flujo o al costo.

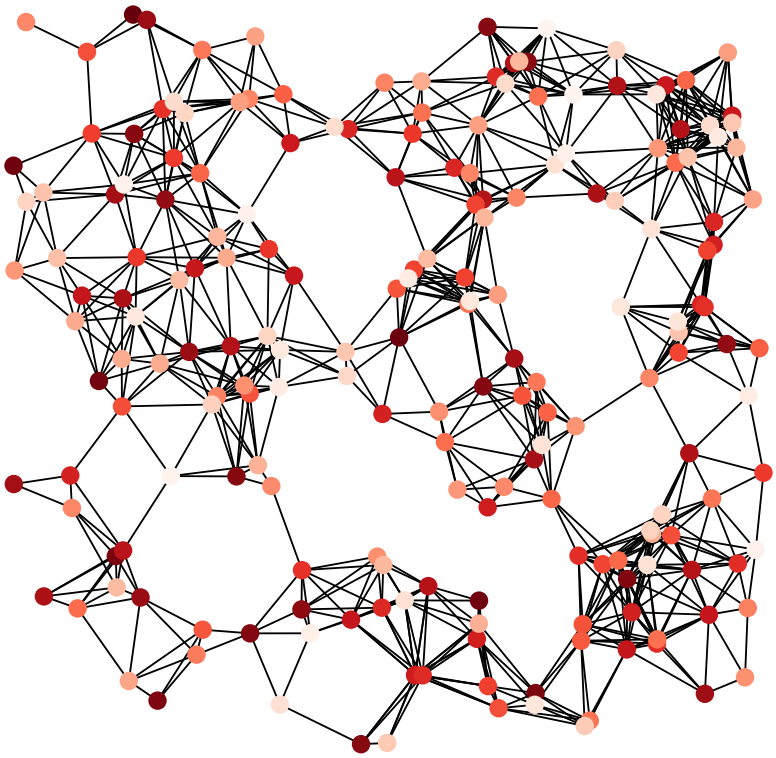


Figura 5: Mejores candidatos a nodos hubs

7. Conclusiones

Se caracterizaron los nodos según su influencia en el flujo, en el costo y por su conectividad. Se analizó la red como un sistema y se definieron nodos críticos en cuanto al flujo y al costo. Finalmente se proponen como hubs una serie de nodos que cumplen con los tres supuestos propuestos inicialmente. Fue posible observar los efectos que provoca en la red eliminar cada nodo. Se propusieron al menos un nodo hub por cada una de las zonas más densas en el grafo.

Referencias

[1] Janet M Six and Ioannis G Tollis. A framework for circular drawings of networks. In *International Symposium on Graph Drawing*, pages 107–116. Springer, 1999.

[2] Yehuda Koren. On spectral graph drawing. In *International Computing and Combinatorics Conference*, pages 496–508. Springer, 2003.

[3] John G Klincewicz. Hub location in backbone/tributary network design: a review. *Location Science*, 6(1-4):307–335, 1998.

[4] Sibel Alumur and Bahar Y Kara. Network hub location problems: The state of the art. *European journal of operational research*, 190(1):1–21, 2008.