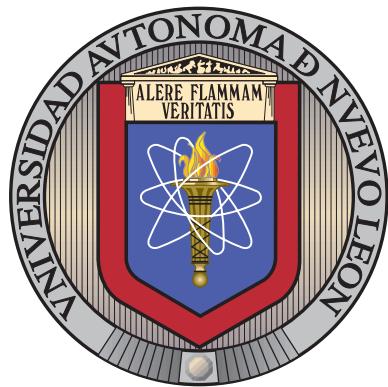


UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA  
POSGRADO EN INGENIERÍA DE SISTEMAS

OPTIMIZACIÓN DE FLUJO EN REDES  
DRA. ELISA SCHAEFFER



CORRECCIÓN DE LAS TAREAS REALIZADAS  
DURANTE EL PERÍODO ENERO–JUNIO DE 2019

POR

MADAY HERNÁNDEZ QUEVEDO

1985280

SAN NICOLÁS DE LOS GARZA, NUEVO LEÓN

JUNIO 2019

## **1. Introducción**

En el portafolio aparece la retroalimentación de la profesora a las tareas de clase, con las correcciones realizadas a las mismas.



Tarea 1:) Representación de redes a través de la teoría de grafos

C Maday Hernández Quevedo) 5280

11 de febrero de 2019

## Introducción

La teoría de grafos tiene aplicación en diversas áreas, debido a que las redes aparecen prácticamente en todas las actividades de nuestra vida diaria: sistemas de comunicaciones, sistemas hidráulicos, circuitos eléctricos y electrónicos, sistemas mecánicos, redes sociales, entre otros.

Como constata Ravindra [1], las redes físicas son quizás las más comunes y mejor identificables. Los sistemas de transporte, cualquiera que sea el medio suelen modelarse en sistemas de distribución y decisiones logísticas complejos, siendo el Problema de transporte un ejemplo típico de investigación de operaciones.

En este problema, un transportista con inventario de mercancías en sus almacenes debe enviar estos productos a centros minoristas geográficamente dispersos, cada uno con una demanda dada del cliente, donde el objetivo es incurrir en los mínimos gastos de transporte posibles. Una de las maneras de resolver este problema es mediante algoritmos computacionales basados en grafos.

Otro ejemplo representativo de uso de grafos son los algoritmos de búsqueda web de Google. Estos se basan en el gráfico WWW, que contiene todas las páginas web como vértices y los hipervínculos como bordes. [2]

La teoría de los grafos también ha sido usada en química, debido a la posibilidad de representar los modelos estructurales mediante diagramas. En un grafo molecular, los vértices representan a los átomos y los lados a los enlaces químicos que conectan ciertas parejas de átomos. [4]

## 1. Grafo simple no dirigido acíclico

La mayoría de los esquemas de líneas de autobuses, tranvías o trenes del transporte público pueden ser representados por grafos simples no dirigidos acíclicos.

A continuación se muestra un sección ochos estaciones de la Línea 2 del Metro de Monterrey, donde las estaciones son los nodos y el tramo de línea entre ellos, los vértices.

Listing 1: Representación con un grafo de ocho estaciones de la Línea 2 del Metro de Monterrey.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 11 14:15:32 2019
@author: Madys
"""

import matplotlib.pyplot as plot
import networkx as nx

G = nx . Graph ()

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

pos = {1:(200, 50), 2:(250,100), 3:(300, 150), 4:(350,200), 5:(400,250),
       6:(450,300),7:(500,350),8:(550,400)}

G.add_edge(1,2)
G.add_edge(2,3)
G.add_edge(3,4)
G.add_edge(4,5)
G.add_edge(5,6)
G.add_edge(6,7)
G.add_edge(7,8)

nx.draw_networkx_nodes(G, pos, node_size=400, node_color='y',
                      node_shape='o')
nx.draw_networkx_edges(G, pos, width=1, alpha=0.8, edge_color='black')

labels = []
labels[1] = r'Sendero'
labels[2] = r'Santiago Tapia'
```

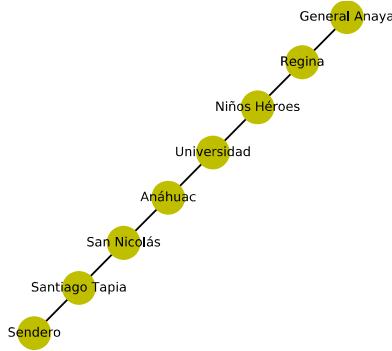


Figura 1: Representación con un grafo de ocho estaciones de la Línea 2 del Metro de Monterrey.

```

    /           \
labels[3] = r'San Nicolás'
labels[4] = r'Anáhuac'
labels[5] = r'Universidad'
labels[6] = r'Niños Héroes'
labels[7] = r'Regina'
labels[8] = r'General Anaya'

nx.draw_networkx_labels(G, pos, labels, font_size=8)
plot.xlim(0,600)
plot.axis('off')
plot.savefig("1.eps")
plot.show()

```

## 2. Grafo simple no dirigido cíclico

Este tipo de grafos es adecuado para representar relaciones comerciales entre empresas al igual que relaciones de parentezco, amistad o romance entre personas. A continuación se muestra un grafo que representa las relaciones de amistad en un grupo de 10 personas. Los vértices son las personas y las aristas, las personas que tienen una relación de amistad.

Listing 2: Relaciones de amistad en un grupo de personas.

```

# -*- coding: utf-8 -*-
"""

```

```

Created on Mon Feb 11 14:15:32 2019
@author: Madys

"""
import matplotlib.pyplot as plot
import networkx as nx

G = nx . Graph ()
pos = {1:(300, 200), 2:(150,100), 3:(250, 300), 4:(350,300),
       5:(450,200), 6:(150,200),7:(370,200),8:(350,100)}

G.add_edge(1,2)
G.add_edge(1,3)
G.add_edge(1,4)
G.add_edge(4,5)
G.add_edge(2,6)
G.add_edge(7,8)
G.add_edge(4,5)
G.add_edge(3,6)
G.add_edge(1,6)
G.add_edge(1,7)

nx.draw_networkx_nodes(G, pos, node_size=400, node_color='y',
                      node_shape='o')
nx.draw_networkx_edges(G, pos, width=1, alpha=0.8, edge_color='black')

labels = []
labels[1] = r'Mario'
labels[2] = r'Betty'
labels[3] = r'Arturo'
labels[4] = r'Carlos'
labels[5] = r'Anna'
labels[6] = r'Jane'
labels[7] = r'Emily'
labels[8] = r'Andrew'

nx.draw_networkx_labels(G, pos, labels, font_size=12)
plot.xlim(0,500)
plot.axis('off')
plot.savefig("2.eps")
plot.show()

```

### 3. Grafo simple no dirigido reflexivo

Un criador de perros posee 6 razas diferentes de estos animales. Con un grafo de este tipo puede representar de cuales de estos animales ha obtenido crías, siendo

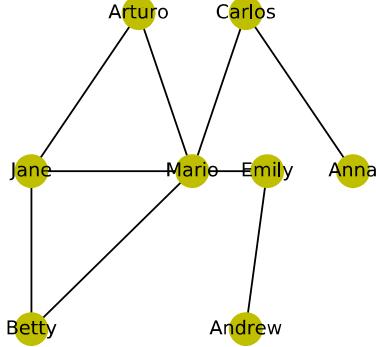


Figura 2: Relaciones de amistad en un grupo de personas.

los vértices las razas de perro y las aristas la unión entre razas que han tenido crías. En el grafo se puede observar que todas las razas excepto la Beagle, que es recién adquirida y aún no se ha reproducido, han tenido crías con su misma raza (vértices en rojo). También se observa que han habido cruzamientos entre bóxer y bulldog y entre pitbull y rottweiler.

Listing 3: Estado de cría de razas de perro.

```

# -*- coding: utf-8 -*-
"""
Created on Mon Feb 11 15:37:20 2019
@author: Madys
"""

import matplotlib.pyplot as plot
import networkx as nx

G = nx . Graph ()

G.add_edge(1,2)
G.add_edge(3,4)
G.add_node(5)
G.add_node(6)
apareados={1,2,3,4,5}
noApareado={6}
pos = {1:(200, 350), 2:(550,350), 3:(650, 220), 4:(400,100),
       5:(150,220),6:(100,100)}

```

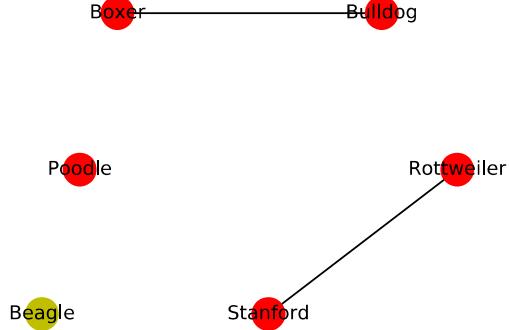


Figura 3: Estado de cría de razas de perro

```

nx.draw_networkx_nodes(G, pos,nodelist=apareados,node_size=400,
                      node_color='r', node_shape='o')
nx.draw_networkx_nodes(G, pos,nodelist=noApareado, node_size=400,
                      node_color='y', node_shape='o')
nx.draw_networkx_edges(G, pos, width=1, alpha=0.8, edge_color='black')

labels = {}
labels[1] = r'Boxer'
labels[2] = r'Bulldog'
labels[3] = r'Rottweiler'
labels[4] = r'Stanford'
labels[5] = r'Poodle'
labels[6] = r'Beagle'

nx.draw_networkx_labels(G, pos, labels, font_size=12)
plot.xlim(20,730)
plot.axis('off')
plot.savefig("3.eps")
plot.show()

```

#### 4. Grafo simple dirigido acíclico

La cadena de propagación de las ITS (Infecciones de Transmisión Sexual) para las cuales no se conoce cura o las que solo afectan al individuo una vez en la vida, pueden representarse mediante grafos simples dirigidos acíclicos. En este

caso cada sujeto es un vértice y la dirección de transmisión de la enfermedad es representada en la arista. Desde la enfermedad se propaga desde la persona portadora hacia el sujeto sano que posteriormente se convierte en portador y contagia a otros sujetos sanos.

Listing 4: Representación de la transmisión de una ITS en un grupo de personas.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 11 14:15:32 2019
@author: Madys
"""

import matplotlib.pyplot as plot
import networkx as nx

G = nx.Graph()
G.add_node(1)
G.add_node(2)
G.add_node(3)
G.add_node(4)
G.add_node(5)
G.add_node(6)
G.add_node(7)
G.add_node(8)
pos = {1:(300, 400), 2:(100,300), 3:(250, 300), 4:(350,300),
       5:(450,200), 6:(150,200),7:(300,200),8:(250,100)}

G.add_edge(1,2)
G.add_edge(1,3)
G.add_edge(1,4)
G.add_edge(4,5)
G.add_edge(2,6)
G.add_edge(7,8)
G.add_edge(4,7)

nx.draw_networkx_nodes(G, pos, node_size=400, node_color='y',
                       node_shape='o')
nx.draw_networkx_edges(G, pos, width=1, alpha=0.8, edge_color='black')

labels = {}
labels[1] = r'Mario'
labels[2] = r'Betty'
labels[3] = r'Arturo'
labels[4] = r'Carlos'
labels[5] = r'Anna'
labels[6] = r'Jane'
labels[7] = r'Emily'
labels[8] = r'Andrew'
```

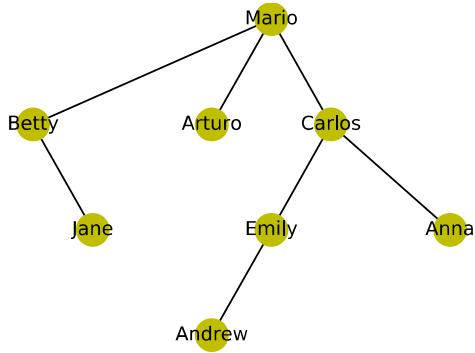


Figura 4: Representación de la transmisión de una ITS en un grupo de personas.

```

nx.draw_networkx_labels(G, pos, labels, font_size=12)
plot.xlim(50,500)
plot.axis('off')
plot.savefig("4.eps")
plot.show()

```

## 5. Grafo simple dirigido cíclico

En ciudades con carreteras estrechas como Santiago de Cuba, en diferentes sectores, las carreteras son de un solo sentido y se representan con grafos de este tipo. Las aristas representan las calles y los vértices son las intersecciones entre al menos dos aristas.

Listing 5: Representación de carreteras de doble sentido en Santiago de Cuba.

```

# -*- coding: utf-8 -*-
"""
Created on Mon Feb 11 12:26:08 2019
@author: Madys
"""

import matplotlib.pyplot as plot
import networkx as nx

G = nx . DiGraph ()

```

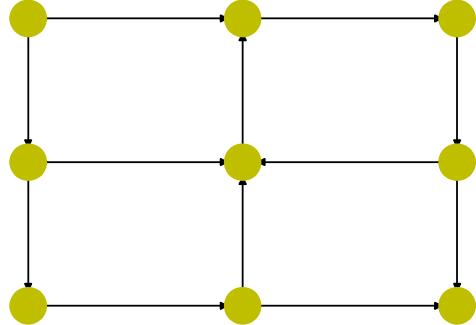


Figura 5: Carreteras de sentido único en Santiago de Cuba.

```

G.add_edge(1,2)
G.add_edge(2,3)
G.add_edge(6,5)
G.add_edge(4,5)
G.add_edge(8,9)
G.add_edge(7,8)
G.add_edge(1,4)
G.add_edge(4,7)
G.add_edge(8,5)
G.add_edge(5,2)
G.add_edge(3,6)
G.add_edge(6,9)

pos = {1:(50, 350), 2:(250,350), 3:(450, 350), 4:(50,200), 5:(250,200),
       6:(450,200),7:(50,50),8:(250,50),9:(450,50)}
nx.draw_networkx_nodes(G, pos, node_size=500, node_color='y',
                       node_shape='o')
nx.draw_networkx_edges(G, pos, width=1, alpha=0.8, edge_color='black')

plot.xlim(0,500)
plot.axis('off')
plot.savefig("5.eps")
plot.show()

```

## 6. Grafo simple dirigido reflexivo

Un grafo en el que cada vértice es una empresa de servicios y las aristas, la unión con cada una de las otras empresas a las que le presta servicios, puede representarse mediante un grafo dirigido reflexivo, pues algunas de estas empresas se brindan servicio a ellas mismas.

Un sitio web pequeño, en el cual se accede a todas sus páginas a través de un menú estático también se puede representar con este tipo de grafo.

Otro ejemplo está dado por la representación de un grupo de personas conectadas a una red social y los perfiles de otras personas que tengan abiertos en el navegador en ese momento. Cada una de esas persona también pudiesen estar mirando su propio perfil. En el grafo propuesto los vértices son las personas y las aristas indican el perfil de qué personas tiene cada uno abierto en el navegador. Puede apreciarse en color rojo que Mayra y Andrew tienen abiertos sus propios perfiles.

Listing 6: Representación de personas mirando perfiles en redes sociales.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 11 14:15:32 2019
@author: Madys
"""

import matplotlib.pyplot as plot
import networkx as nx

G = nx . DiGraph ()

G.add_edge(1,2)
G.add_edge(1,3)
G.add_edge(1,5)
G.add_edge(2,3)
G.add_edge(3,5)
G.add_edge(4,5)
node1 = {1,2}
node2 = {3,4,5}

pos = {1:(200, 350), 2:(550,350), 3:(650, 220), 4:(400,100), 5:(150,220)}

nx.draw_networkx_nodes(G, pos, nodelist=node1,node_size=400,
                      node_color='r', node_shape='o')
nx.draw_networkx_nodes(G, pos, nodelist=node2,node_size=400,
                      node_color='y', node_shape='o')
nx.draw_networkx_edges(G, pos,width=1, alpha=0.8, edge_color='black')

labels = {}
```

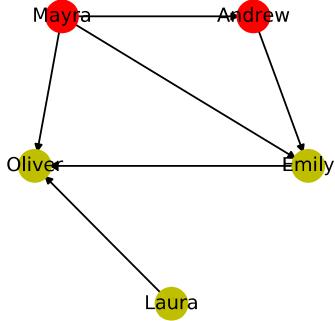


Figura 6: Representación de personas mirando perfiles en redes sociales.

```

labels[1] = r'Mayra'
labels[2] = r'Andrew'
labels[3] = r'Emily'
labels[4] = r'Laura'
labels[5] = r'Oliver'

nx.draw_networkx_labels(G, pos, labels, font_size=12)
plot.xlim(20,1000)
plot.axis('off')
plot.savefig("6.eps")
plot.show()

```

## 7. Multigrafo no dirigido acíclico

La ruta entre Santiago de Cuba y Holguín puede representarse como un multigrafo no dirigido cíclico. En un pueblo llamado Caballería la carretera principal se bifurca, pudiendo continuar por la ruta principal hasta Banes, o por el camino alternativo de Caballería, que, aunque es más largo, puede recorrerse en menor tiempo a causa del poco tránsito.

En la figura los vértices representan las cabeceras municipales, que están simplemente enumeradas por no ser de interés para el ejemplo mencionado.

Listing 7: Posible ruta entre Santiago de Cuba y Holguín.

```
# -*- coding: utf-8 -*-

```

```

"""
Created on Mon Feb 11 14:15:32 2019
@author: Madys
"""

import matplotlib.pyplot as plot
import networkx as nx

G = nx . MultiGraph ()

G.add_edge(1,2)
G.add_edge(2,3)
G.add_edge(3,4)
G.add_edge(4,5)
G.add_edge(5,6, weight=3)
G.add_edge(5,6,weight=2)
G.add_edge(6,7)
black=[(1,2),(2,3),(3,4),(4,5),(6,7)]
blue=[(5,6)]
red=[(5,6)]

pos = {1:(100, 100), 2:(200,180), 3:(280, 250), 4:(320,300),
       5:(450,380),6:(500,480), 7:(570,580)}

nx.draw_networkx_nodes(G, pos ,node_size=400, node_color='y',
                      node_shape='o')
nx.draw_networkx_edges(G, pos,edgelist=black,width=1,edge_color='black',
                       alpha=0.8)
nx.draw_networkx_edges(G, pos, edgelist=blue,width=6, alpha=0.5,
                      edge_color='b', style='dashed')
nx.draw_networkx_edges(G, pos, edgelist=red,width=4, alpha=0.5,
                      edge_color='r')
nx.draw_networkx_labels(G, pos, labels, font_size=12)

labels = {}
labels[1] = r'Santiago'
labels[2] = r'1'
labels[3] = r'2'
labels[4] = r'3'
labels[5] = r'Caballera'
labels[6] = r'Caballera'
labels[7] = r'Holgun'

plot.xlim(20,800)
plot.axis('off')
plot.savefig("7.eps")
plot.show()

```

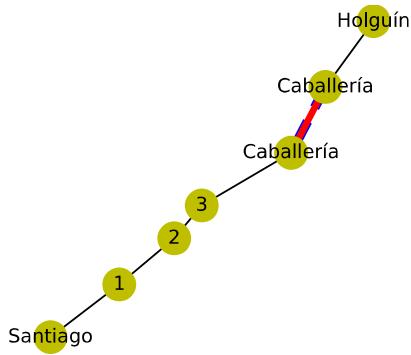


Figura 7: Posible ruta entre Santiago de Cuba y Holguín

## 8. Multigrafo no dirigido cíclico

En la presa El cacao, ubicada en el Municipio Cotorro, Ciudad de La Habana, se realizó un proyecto para favorecer la agricultura. Al rededor de la presa se construyeron una serie de canales para mantener el suelo húmedo durante todo el año. Este sistema puede ser representado mediante el grafo mostrado a continuación, el cual los vértices representan la unión entre uno o más canales y las aristas, cada uno de los canales.

Listing 8: Canales para irrigación del suelo creados en la presa El cacao.

```

# -*- coding: utf-8 -*-
"""
Created on Mon Feb 11 12:26:08 2019
@author: Madys
"""

import matplotlib.pyplot as plot
import networkx as nx

G = nx . MultiGraph ()

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

```

```

G.add_edge(1,4, weight=3)
G.add_edge(4,5, weight=5)
G.add_edge(4,5, weight=3)
G.add_edge(1,6, weight=3)
G.add_edge(1,6,weight=2)
G.add_edge(6,7, weight=1)
G.add_edge(6,7, weight=4)
G.add_edge(1,8, weight=5)
G.add_edge(1,8, weight=4)
G.add_edge(1,8, weight=2)
G.add_edge(8,9, weight=5)
G.add_edge(8,9, weight=4)

blue=[(1,2),(2,3),(1,4),(4,5),(1,6),(6,7),(1,8),(8,9)]
red=[(1,2),(2,3),(1,4),(4,5),(1,6),(6,7),(1,8),(8,9)]
green=[(1,2),(2,3),(1,8)]


pos = {1:(400, 400), 2:(300,300), 3:(100, 100), 4:(500,500),
       5:(650,650),6:(250,500), 7:(100
       ,700), 8:(500, 300), 9:(700,300)}


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

nx.draw_networkx_edges(G, pos, edgelist=blue,width=6, alpha=0.5,
                      edge_color='b', style='dashed')
nx.draw_networkx_edges(G, pos, edgelist=green,width=5, alpha=0.5,
                      edge_color='g')
nx.draw_networkx_edges(G, pos, edgelist=red,width=4, alpha=0.5,
                      edge_color='r')
labels = {}
labels[1] = r'$Presa$'

nx.draw_networkx_labels(G, pos, labels, font_size=12)

plot.xlim(20,800)
plot.axis('off')
plot.savefig("8.eps")
plot.show()

```

## 9. Multigrafo no dirigido reflexivo

En la Universidad de Oriente se quiere realizar un concurso de habilidades entre las carreras de Ingeniería en Electrónica, Ingeniería en Automática, Licenciatura en Física, Ingeniería en Informática y Licenciatura en Matemática-Cibernética. Las habilidades a evaluar serán matemática y programación.

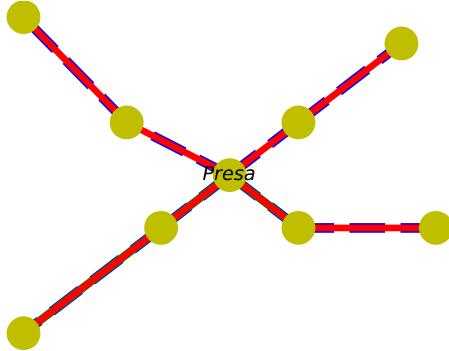


Figura 8: RCanales para irrigación del suelo creados en la presa El cacao.

El siguiente grafo muestra como está organizado el concurso. Cada vértice representa a los estudiantes que estudian una de las carreras y las aristas, con qué estudiantes pueden participar en cada habilidad. Todos los vértices son de color rojo porque todos los estudiantes pueden competir con estudiantes de su misma carrera.

Los vértices rojos unen a los que pueden concursar entre sí en matemáticas y los azules, los que pueden concursar en programación.

Listing 9: Concurso de habilidades.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 11 14:15:32 2019
@author: Madys
"""

import matplotlib.pyplot as plot
import networkx as nx

G = nx . MultiGraph ()

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

blue=[(1,2),(4,5)]
```

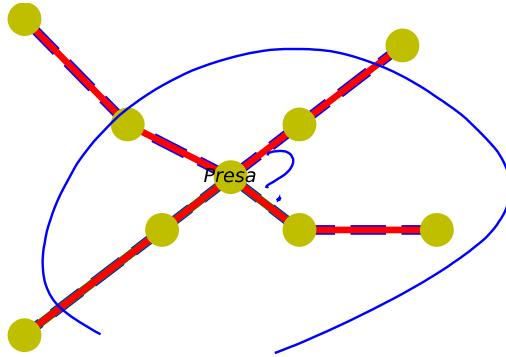


Figura 9: Concurso de habilidades.

```

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

pos = {1:(200, 100), 2:(100,400), 3:(200, 700), 4:(500,700), 5:(650,400)}

nx.draw_networkx_nodes(G, pos ,node_size=400, node_color='r',
                      node_shape='o')
nx.draw_networkx_edges(G, pos, edgelist=blue, width=6, alpha=0.5,
                      edge_color='b', style='dashed')

nx.draw_networkx_edges(G, pos, edgelist=red, width=4, alpha=0.5,
                      edge_color='r')
labels = {}
labels[1] = r'Automtica'
labels[2] = r'Elctrica'
labels[3] = r'Fsica'
labels[4] = r'Informtica'
labels[5] = r'Ciberntica'

nx.draw_networkx_labels(G, pos, labels, font_size=12)

plot.xlim(20,800)
plot.axis('off')
plot.savefig("9.eps")
plot.show()

```

## 10. Multigrafo dirigido acíclico

Estos grafos pueden utilizarse para representar la cuenca hidrográfica de un río en su flujo hacia el mar. Los vértices representan los puntos en los que al menos dos ramificaciones del delta confluyen o se separan.

Listing 10: Representación del delta de un río en su transcurso hacia el mar.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 11 16:44:18 2019
@author: Madys
"""

import matplotlib.pyplot as plot
import networkx as nx

G = nx . MultiDiGraph ()

G.add_edge(1,2, weight=3)
G.add_edge(1,2, weight=5)
G.add_edge(1,2, weight=6)
G.add_edge(2,3, weight=4)
G.add_edge(2,5, weight=3)
G.add_edge(2,5, weight=5)
G.add_edge(1,4, weight=4)
G.add_edge(1,4, weight=3)
G.add_edge(4,5, weight=5)
G.add_edge(4,5, weight=3)
G.add_edge(1,6, weight=3)
G.add_edge(1,6,weight=2)
G.add_edge(6,7, weight=1)
G.add_edge(6,7, weight=4)
G.add_edge(1,8, weight=5)
G.add_edge(1,8, weight=4)
G.add_edge(1,8, weight=2)
G.add_edge(8,9, weight=5)
G.add_edge(8,9, weight=4)

blue=[(1,2),(2,3),(2,5),(1,4),(4,5),(1,6),(6,7),(1,8),(8,9)]
red=[(1,2),(2,5),(1,4),(4,5),(1,6),(6,7),(1,8),(8,9)]
green=[(1,2),(2,5),(1,8)]

pos = {1:(400, 700), 2:(300,300), 3:(150, 100), 4:(400,200),
       5:(300,100),6:(250,500), 7:(100
       ,100), 8:(500, 300), 9:(700,100) }

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

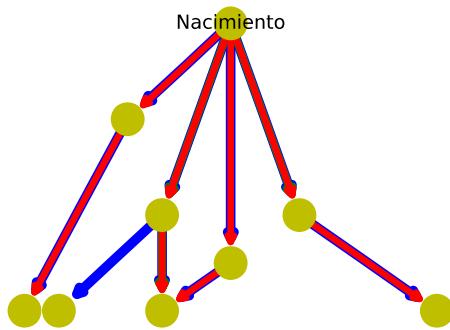


Figura 10: Representación del delta de un río en su transcurso hacia el mar.

```

nx.draw_networkx_edges(G, pos, edgelist=blue, width=6, alpha=0.5,
edge_color='b', style='dashed')
nx.draw_networkx_edges(G, pos, edgelist=green, width=5, alpha=0.5,
edge_color='g')
nx.draw_networkx_edges(G, pos, edgelist=red, width=4, alpha=0.5,
edge_color='r')
labels = {}
labels[1] = r'Nacimiento'

nx.draw_networkx_labels(G, pos, labels, font_size=12)

plot.xlim(20,800)
plot.axis('off')
plot.savefig("10.eps")
plot.show()

```

## 11. Multigrafo dirigido cíclico

Este tipo de grafo es especialmente útil para representar viajes por lugares a los que se puede llegar mediante diferentes vías, regresando luego al punto de origen. Un ejemplo concreto de esto sería un recorrido vacacional por varias ciudades. Partiendo de la ciudad A hay tres posibles vías para llegar a la ciudad B, en bus, en auto de alquiler o en tren, cada uno de estos medios de transporte representaría una arista diferente para unir los vértices. Así sucesivamente, se tienen en cuenta las posibles vías de transporte hacia las distintas ciudades hasta

\$a\$

18

\$b\$

completar el recorrido y regresar a casa.

Listing 11: Representación de las posibles maneras de elegir la ruta de un circuito vacacional.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 11 17:00:30 2019
@author: Madys
"""

import matplotlib.pyplot as plot
import networkx as nx

G = nx . MultiDiGraph ()

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

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

pos = {1:(400, 700), 2:(700,600), 3:(550, 400), 4:(400,200),
       5:(300,100),6:(150,200), 7:(100
       ,400)}

nx.draw_networkx_nodes(G, pos ,node_size=400, node_color='y',
                      node_shape='o')
nx.draw_networkx_edges(G, pos, edgelist=blue,width=6, alpha=0.5,
                      edge_color='b', style='dashed')
nx.draw_networkx_edges(G, pos, edgelist=green,width=5, alpha=0.5,
                      edge_color='g')
nx.draw_networkx_edges(G, pos, edgelist=red,width=4, alpha=0.5,
                      edge_color='r')

labels = []
labels[1] = r'Casa'
labels[2] = r'A'
labels[3] = r'B'
```

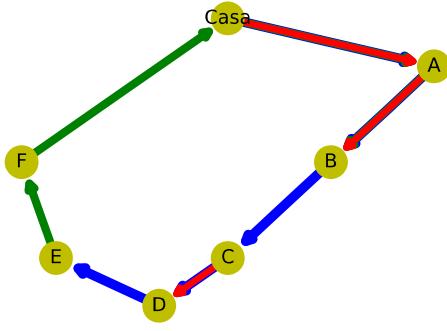


Figura 11: Representación de las posibles maneras de elegir la ruta de un circuito vacacional.

```

labels[4] = r'C'
labels[5] = r'D'
labels[6] = r'E'
labels[7] = r'F'

nx.draw_networkx_labels(G, pos, labels, font_size=12)
plot.xlim(20,800)
plot.axis('off')
plot.savefig("11.eps")
plot.show()

```

## 12. $\alpha$ multigrafo dirigido reflexivo

González-Cervantes [3] empleó teoría de grafos para representar el potencial eléctrico en el corazón, demostrando que se pueden incorporar las leyes fisiológicas involucradas. Cada uno de los vértices representa uno de los puntos principales que generan los impulsos eléctricos y que llevan la electricidad a cada parte del corazón, las aristas describen el valor máximo de voltaje y su duración en tiempo que descarga cada vértice. Además, puede proporcionar información con respecto al potencial eléctrico por zonas para una mejor localización. En la imagen se muestra la representación con grafos del intervalo PR.

Listing 12: Representación con grafos del intervalo PR.

newpage

```

Created on Mon Feb 11 17:00:30 2019
@author: Madys
"""

import matplotlib.pyplot as plot
import networkx as nx

G = nx . MultiDiGraph ()

G.add_edge(1,2)
G.add_edge(2,3)
G.add_edge(3,4)
G.add_edge(3,5)
G.add_edge(1,6,weight=3)
G.add_edge(1,6,weight=1)
blue=[(1,6)]
G.add_edge(2,6)
node1 = {6}

pos = {1:(50, 350), 2:(250,350), 3:(450, 350), 4:(600,350), 5:(550,340),
       6:(450,330)}
nx.draw_networkx_nodes(G, pos, node_size=500, node_color='y',
                      node_shape='o')
nx.draw_networkx_edges(G, pos, width=1, alpha=0.8, edge_color='black')
nx.draw_networkx_nodes(G, pos, nodelist=node1,node_size=400,
                      node_color='r', node_shape='o')
nx.draw_networkx_edges(G, pos, edgelist=blue,width=6, alpha=0.5,
                      edge_color='b', style='dashed')

labels = {}
labels[1] = r'v1'
labels[2] = r'v2'
labels[3] = r'v3'
labels[4] = r'e21'
labels[5] = r'e14'
labels[6] = r'v6'

nx.draw_networkx_labels(G, pos, labels, font_size=12)
plot.xlim(20,800)
plot.axis('off')
plot.savefig("12.eps")
plot.show()

```

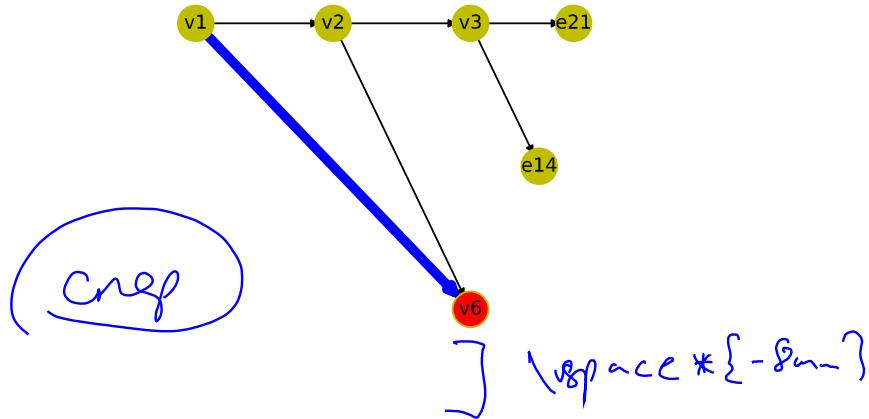


Figura 12: Representación con grafos del intervalo PR.

## Referencias

- [1] Ravindra K Ahuja. *Network flows: theory, algorithms, and applications*. Pearson Education, 2017.
- [2] Fan Chung. Graph theory in the information age. *Notices of the AMS*, 57(6):726–732, 2010.
- [3] Natalia González-Cervantes, Aurora Espinoza-Valdez, and Ricardo Salido-Ruiz. Potencial eléctrico en el corazón: Representación mediante un grafo. *ReCIBE*, 5(3), 2016.
- [4] Amador Menéndez Velázquez. Una breve introducción a la teoría de grafos. *Suma*, 28:11–26, 1998.

# Tarea 1

5280

1 de junio de 2019

## Introducción

La teoría de grafos tiene aplicación en diversas áreas, debido a que las redes aparecen prácticamente en todas las actividades de nuestra vida diaria: sistemas de comunicaciones, sistemas hidráulicos, circuitos eléctricos y electrónicos, sistemas mecánicos, redes sociales, entre otros.

Como constata Ravindra [1], las redes físicas son quizás las más comunes y mejor identificables. Los sistemas de transporte, cualquiera que sea el medio suelen modelarse en sistemas de distribución y decisiones logísticas complejos, siendo el Problema de transporte un ejemplo típico de investigación de operaciones.

En este problema, un transportista con inventario de mercancías en sus almacenes debe enviar estos productos a centros minoristas geográficamente dispersos, cada uno con una demanda dada del cliente, donde el objetivo es incurrir en los mínimos gastos de transporte posibles. Una de las maneras de resolver este problemas es mediante algoritmos computacionales basados en grafos.

Otro ejemplo representativo de uso de grafos son los algoritmos de búsqueda web de Google. Estos se basan en el gráfico WWW, que contiene todas las páginas web como vértices y los hipervínculos como bordes [2].

La teoría de los grafos también ha sido usada en en química, debido a la posibilidad de representar los modelos estructurales mediante diagramas. En un grafo molecular, los vértices representan a los átomos y los lados a los enlaces químicos que conectan ciertas parejas de átomos [4].

## 1. Grafo simple no dirigido acíclico

La mayoría de los esquemas de líneas de autobuses, tranvías o trenes del transporte público pueden ser representados por grafos simples no dirigidos acíclicos.

En la figura 1 (página 3) se muestra un sección ochos estaciones de la Línea 2 del Metro de Monterrey, donde las estaciones son los nodos y el tramo de línea entre ellos, los vértices.

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Mon Feb 11 14:15:32 2019
4 @author: Madys
5 """
6
7 import matplotlib.pyplot as plot
8 import networkx as nx
```

```

9 G = nx . Graph ()
10
11 G.add_node(1)
12 G.add_node(2)
13 G.add_node(3)
14 G.add_node(4)
15 G.add_node(5)
16 G.add_node(6)
17 G.add_node(7)
18 G.add_node(8)
19 G.add_node(8)
20 pos = {1:(200, 50), 2:(250,100), 3:(300, 150), 4:(350,200), 5:(400,250), 6:(450,300),
21 ,7:(500,350),8:(550,400)}
22
23 G.add_edge(1,2)
24 G.add_edge(2,3)
25 G.add_edge(3,4)
26 G.add_edge(4,5)
27 G.add_edge(5,6)
28 G.add_edge(6,7)
29 G.add_edge(7,8)
30
31 nx.draw_networkx_nodes(G, pos, node_size=400, node_color='y', node_shape='o')
32 nx.draw_networkx_edges(G, pos, width=1, alpha=0.8, edge_color='black')
33
34 labels = {}
35 labels[1] = r'Sendero'
36 labels[2] = r'Santiago Tapia'
37 labels[3] = r'San Nicolas'
38 labels[4] = r'Anahuac'
39 labels[5] = r'Universidad'
40 labels[6] = r'Ninos Heroes'
41 labels[7] = r'Regina'
42 labels[8] = r'General Anaya'
43
44 nx.draw_networkx_labels(G, pos, labels, font_size=8)
45 plot.xlim(0,600)
46 plot.axis('off')
47 plot.savefig("1.eps")
48 plot.show()

```

1.py

## 2. Grafo simple no dirigido cíclico

Este tipo de grafos es adecuado para representar relaciones comerciales entre empresas al igual que relaciones de parentezco, amistad o romance entre personas. En la figura 2 (página 4) se muestra un grafo que representa las relaciones de amistad en un grupo de diez personas. Los vértices son las personas y las aristas, las personas que tienen una relación de amistad.

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Mon Feb 11 14:15:32 2019
4 @author: Madys
5 """
6
7 import matplotlib.pyplot as plot
8 import networkx as nx
9
10 G = nx . Graph ()

```

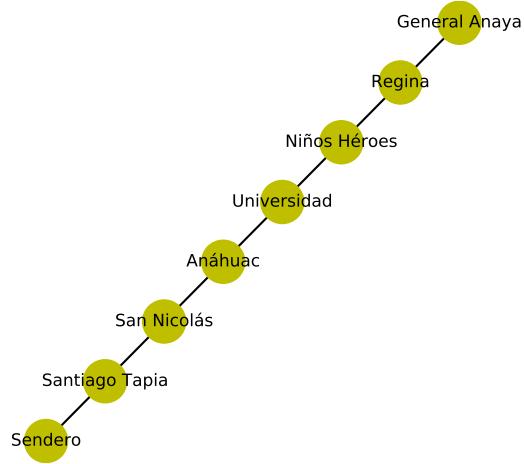


Figura 1: Representación con un grafo de ocho estaciones de la Línea 2 del Metro de Monterrey.

```

11 pos = {1:(300, 200), 2:(150,100), 3:(250, 300), 4:(350,300), 5:(450,200),
12 6:(150,200),7:(370,200),8:(350,100)}
13 G.add_edge(1,2)
14 G.add_edge(1,3)
15 G.add_edge(1,4)
16 G.add_edge(4,5)
17 G.add_edge(2,6)
18 G.add_edge(7,8)
19 G.add_edge(4,5)
20 G.add_edge(3,6)
21 G.add_edge(1,6)
22 G.add_edge(1,7)
23
24 nx.draw_networkx_nodes(G, pos, node_size=400, node_color='y', node_shape='o')
25 nx.draw_networkx_edges(G, pos, width=1, alpha=0.8, edge_color='black')
26
27 labels = []
28 labels[1] = r'Mario'
29 labels[2] = r'Betty'
30 labels[3] = r'Arturo'
31 labels[4] = r'Carlos'
32 labels[5] = r'Anna'
33 labels[6] = r'Jane'
34 labels[7] = r'Emily'
35 labels[8] = r'Andrew'
36
37 nx.draw_networkx_labels(G, pos, labels, font_size=12)
38 plot.xlim(0,500)
39 plot.axis('off')
40 plot.savefig("2.eps")
41 plot.show()

```

2.py

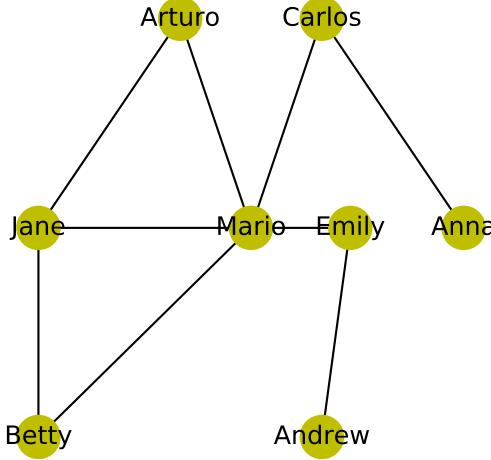


Figura 2: Relaciones de amistad en un grupo de personas.

### 3. Grafo simple no dirigido reflexivo

Un criador de perros posee seis razas diferentes de estos animales. Con un grafo de este tipo puede representar de cuales de estos animales ha obtenido crías, siendo los vértices las razas de perro y las aristas la unión entre razas que han tenido crías. En la 3 (página 5) se puede observar que todas las razas excepto la Beagle, que es recién adquirida y aún no se ha reproducidos, han tenido crías con su misma raza (vértices en rojo). También se observa que han habido cruzamientos entre bóxer y bulldog y entre pitbull y rottweiler.

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Mon Feb 11 15:37:20 2019
4 @author: Madys
5 """
6
7 import matplotlib.pyplot as plot
8 import networkx as nx
9
10 G = nx . Graph ()
11
12 G.add_edge(1 ,2)
13 G.add_edge(3 ,4)
14 G.add_node(5)
15 G.add_node(6)
16 apareados={1,2,3,4,5}
17 noApareado={6}
18 pos = {1:(200, 350), 2:(550,350), 3:(650, 220), 4:(400,100), 5:(150,220),6:(100,100)}
19
20 nx.draw_networkx_nodes(G, pos ,nodelist=apareados ,node_size=400, node_color='r' ,
21 node_shape='o')
22 nx.draw_networkx_nodes(G, pos ,nodelist=noApareado , node_size=400, node_color='y' ,
23 node_shape='o')
24 nx.draw_networkx_edges(G, pos ,width=1, alpha=0.8, edge_color='black')
25 labels = {}
labels [1] = r 'Boxer',

```

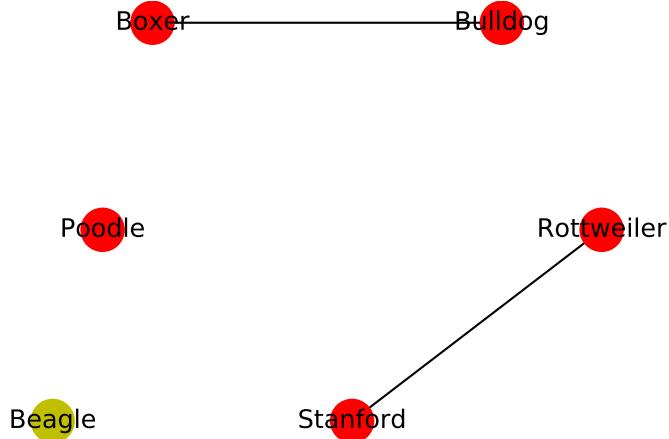


Figura 3: Estado de cría de razas de perro

```

26 labels[2] = r'Bulldog'
27 labels[3] = r'Rottweiler'
28 labels[4] = r'Stanford'
29 labels[5] = r'Poodle'
30 labels[6] = r'Beagle'
31
32 nx.draw_networkx_labels(G, pos, labels, font_size=12)
33 plot.xlim(20,730)
34 plot.axis('off')
35 plot.savefig("3.eps")
36 plot.show()

```

3.py

#### 4. Grafo simple dirigido acíclico

La cadena de propagación de las ITS (Infecciones de Transmisión Sexual) para las cuales no se conoce cura o las que solo afectan al individuo una vez en la vida, pueden representarse mediante grafos simples dirigidos acíclicos. En este caso cada sujeto es un vértice y la dirección de transmisión de la enfermedad es representada en la arista. Desde la enfermedad se propaga desde la persona portadora hacia el sujeto sano que posteriormente se convierte en portador y contagia a otros sujetos sanos. Esto puede observarse en el grafo de la figura 4 (página 6)

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Mon Feb 11 14:15:32 2019
4 @author: Madys
5
6
7 import matplotlib.pyplot as plot
8 import networkx as nx
9
10 G = nx . Graph ()
11 G.add_node(1)

```

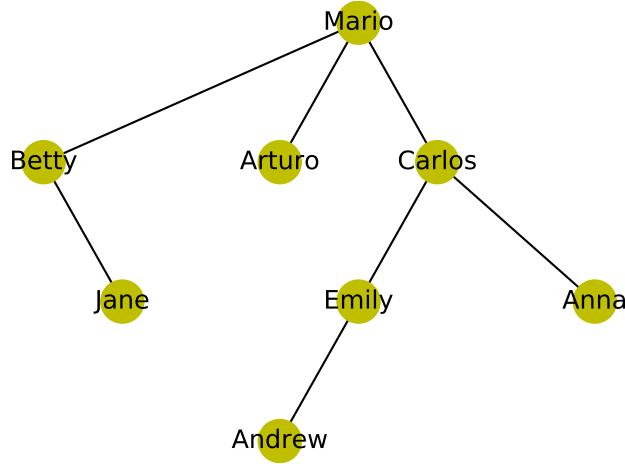


Figura 4: Representación de la transmisión de una ITS en un grupo de personas.

```

12 G.add_node(2)
13 G.add_node(3)
14 G.add_node(4)
15 G.add_node(5)
16 G.add_node(6)
17 G.add_node(7)
18 G.add_node(8)
19 pos = {1:(300, 400), 2:(100,300), 3:(250, 300), 4:(350,300), 5:(450,200),
6:(150,200),7:(300,200),8:(250,100)}
20
21 G.add_edge(1,2)
22 G.add_edge(1,3)
23 G.add_edge(1,4)
24 G.add_edge(4,5)
25 G.add_edge(2,6)
26 G.add_edge(7,8)
27 G.add_edge(4,7)
28
29 nx.draw_networkx_nodes(G, pos, node_size=400, node_color='y', node_shape='o')
30 nx.draw_networkx_edges(G, pos, width=1, alpha=0.8, edge_color='black')
31
32 labels = {}
33 labels[1] = r'Mario'
34 labels[2] = r'Betty'
35 labels[3] = r'Arturo'
36 labels[4] = r'Carlos'
37 labels[5] = r'Anna'
38 labels[6] = r'Jane'
39 labels[7] = r'Emily'
40 labels[8] = r'Andrew'
41
42 nx.draw_networkx_labels(G, pos, labels, font_size=12)
43 plot.xlim(50,500)
44 plot.axis('off')
45 plot.savefig("4.eps")
46 plot.show()

```

4.py

## 5. Grafo simple dirigido cíclico

En ciudades con carreteras estrechas como Santiago de Cuba, en diferentes sectores, las carreteras son de un solo sentido y se representan con grafos de este tipo. Las aristas representan las calles y los vértices son las intersecciones entre al menos dos aristas. Esto puede observarse en la figura 5 (página 8)

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Mon Feb 11 12:26:08 2019
4 @author: Madys
5 """
6 import matplotlib.pyplot as plot
7 import networkx as nx
8
9 G = nx . DiGraph ()
10
11 G.add_edge(1 ,2)
12 G.add_edge(2 ,3)
13 G.add_edge(6 ,5)
14 G.add_edge(4 ,5)
15 G.add_edge(8 ,9)
16 G.add_edge(7 ,8)
17 G.add_edge(1 ,4)
18 G.add_edge(4 ,7)
19 G.add_edge(8 ,5)
20 G.add_edge(5 ,2)
21 G.add_edge(3 ,6)
22 G.add_edge(6 ,9)
23
24 pos = {1:(50 , 350) , 2:(250 ,350) , 3:(450 , 350) , 4:(50 ,200) , 5:(250 ,200) , 6:(450 ,200)
25 ,7:(50 ,50) ,8:(250 ,50) ,9:(450 ,50)}
26 nx.draw_networkx_nodes(G, pos , node_size=500, node_color='y' , node_shape='o')
27 nx.draw_networkx_edges(G, pos , width=1, alpha=0.8, edge_color='black')
28
29 plot.xlim(0,500)
30 plot.axis('off')
31 plot.savefig("5.eps")
32 plot.show()
```

5.py

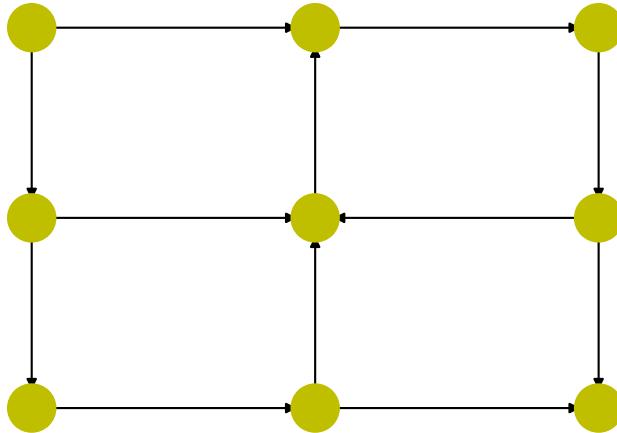


Figura 5: Carreteras de sentido único en Santiago de Cuba.

## 6. Grafo simple dirigido reflexivo

Un grafo en el que cada vértice es una empresa de servicios y las aristas, la unión con cada una de las otras empresas a las que le presta servicios, puede representarse mediante un grafo dirigido reflexivo, pues algunas de estas empresas se brindan servicio a ellas mismas.

Un sitio web pequeño, en el cual se accede a todas sus páginas a través de un menú estático también se puede representar con este tipo de grafo.

Otro ejemplo está dado por la representación de un grupo de personas conectadas a una red social y los perfiles de otras personas que tengan abiertos en el navegador en ese momento. Cada una de esas persona también pudiesen estar mirando su propio perfil. En el grafo mostrado en la figura 6 (página 9) los vértices son las personas y las aristas indican el perfil de qué personas tiene cada uno abierto en el navegador. Puede apreciarse en color rojo que Mayra y Andrew tienen abiertos sus propios perfiles.

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Mon Feb 11 14:15:32 2019
4 @author: Madys
5 """
6
7 import matplotlib.pyplot as plot
8 import networkx as nx
9
10 G = nx . DiGraph ()
11
12 G.add_edge(1 ,2)
13 G.add_edge(1 ,3)
14 G.add_edge(1 ,5)
15 G.add_edge(2 ,3)
16 G.add_edge(3 ,5)
17 G.add_edge(4 ,5)
18 node1 = {1 ,2}
19 node2 = {3 ,4 ,5}
20

```

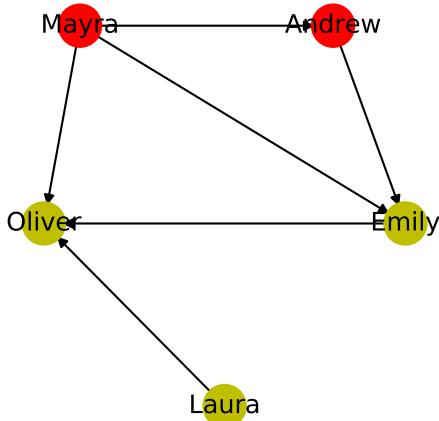


Figura 6: Representación de personas mirando perfiles en redes sociales.

```

21 pos = {1:(200, 350), 2:(550,350), 3:(650, 220), 4:(400,100), 5:(150,220)}
22 nx.draw_networkx_nodes(G, pos, nodelist=node1, node_size=400, node_color='r',
23     node_shape='o')
24 nx.draw_networkx_nodes(G, pos, nodelist=node2, node_size=400, node_color='y',
25     node_shape='o')
26 nx.draw_networkx_edges(G, pos, width=1, alpha=0.8, edge_color='black')
27 labels = {}
28 labels[1] = r'Mayra'
29 labels[2] = r'Andrew'
30 labels[3] = r'Emily'
31 labels[4] = r'Laura'
32 labels[5] = r'Oliver'
33
34 nx.draw_networkx_labels(G, pos, labels, font_size=12)
35 plot.xlim(20,1000)
36 plot.axis('off')
37 plot.savefig("6.eps")
38 plot.show()

```

6.py

## 7. Multigrafo no dirigido acíclico

La ruta entre Santiago de Cuba y Holguín puede representarse como un multigrafo no dirigido cíclico. En un pueblo llamado Caballería la carretera principal se bifurca, pudiendo continuar por la ruta principal hasta Banes, o por el camino alternativo de Caballería, que, aunque es más largo, puede recorrerse en menor tiempo a causa del poco tránsito.

En la figura 7 (página 11) los vértices representan las cabeceras municipales, que están simplemente enumeradas por no ser de interés para el ejemplo mencionado.

```

1 # -*- coding: utf-8 -*-

```

```

2 """
3 Created on Mon Feb 11 14:15:32 2019
4 @author: Madys
5 """
6 import matplotlib.pyplot as plot
7 import networkx as nx
8
9 G = nx . MultiGraph ()
10
11
12 G.add_edge(1 ,2)
13 G.add_edge(2 ,3)
14 G.add_edge(3 ,4)
15 G.add_edge(4 ,5)
16 G.add_edge(5 ,6 , weight=3)
17 G.add_edge(5 ,6 , weight=2)
18 G.add_edge(6 ,7)
19 black=[(1 ,2) ,(2 ,3) ,(3 ,4) ,(4 ,5) ,(6 ,7) ]
20 blue=[(5 ,6)]
21 red=[(5 ,6)]
22
23 pos = {1:(100 , 100) , 2:(200 ,180) , 3:(280 , 250) , 4:(320 ,300) , 5:(450 ,380) ,6:(500 ,480)
24 , 7:(570 ,580)}
25
26 nx.draw_networkx_nodes(G, pos ,node_size=400, node_color='y' , node_shape='o')
27 nx.draw_networkx_edges(G, pos ,edgelist=black ,width=1,edge_color='black' , alpha=0.8)
28 nx.draw_networkx_edges(G, pos , edgelist=blue ,width=6, alpha=0.5,
29 edge_color='b' , style='dashed')
30 nx.draw_networkx_edges(G, pos , edgelist=red ,width=4, alpha=0.5,
31 edge_color='r')
32 nx.draw_networkx_labels(G, pos , labels , font_size=12)
33
34 labels = {}
35 labels [1] = r 'Santiago'
36 labels [2] = r '1'
37 labels [3] = r '2'
38 labels [4] = r '3'
39 labels [5] = r 'Caballeria'
40 labels [6] = r 'Caballeria'
41 labels [7] = r 'Holguin'
42
43 plot.xlim(20 ,800)
44 plot.axis('off')
45 plot.savefig("7.eps")
46 plot.show()

```

7.py

## 8. Multigrafo no dirigido cíclico

En la presa El cacao, ubicada en el Municipio Cotorro, Ciudad de La Habana, se realizó un proyecto para favorecer la agricultura. Al rededor de la presa se construyeron una serie de canales para mantener el suelo húmedo durante todo el año. Este sistema puede ser representado mediante el grafo mostrado en la figura 8 (página 12). Los vértices representan la unión entre uno o más canales y las aristas, cada uno de los canales.

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Mon Feb 11 12:26:08 2019
4 @author: Madys

```

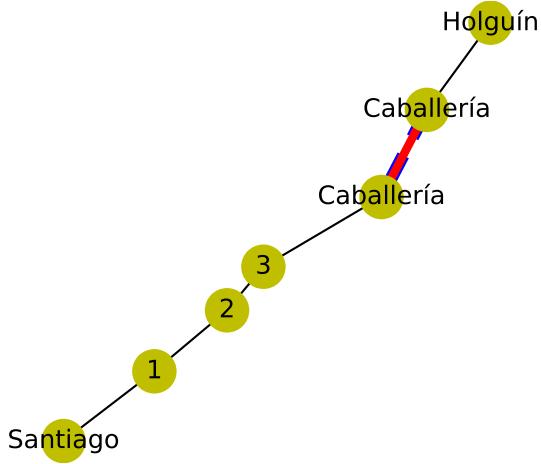


Figura 7: Posible ruta entre Santiago de Cuba y Holguín

```

5 """
6 import matplotlib.pyplot as plot
7 import networkx as nx
8
9 G = nx . MultiGraph ()
10
11 G.add_edge(1 ,2 , weight=3)
12 G.add_edge(1 ,2 , weight=5)
13 G.add_edge(1 ,2 , weight=6)
14 G.add_edge(2 ,3 , weight=4)
15 G.add_edge(2 ,3 , weight=3)
16 G.add_edge(2 ,3 , weight=5)
17 G.add_edge(1 ,4 , weight=4)
18 G.add_edge(1 ,4 , weight=3)
19 G.add_edge(4 ,5 , weight=5)
20 G.add_edge(4 ,5 , weight=3)
21 G.add_edge(1 ,6 , weight=3)
22 G.add_edge(1 ,6 , weight=2)
23 G.add_edge(6 ,7 , weight=1)
24 G.add_edge(6 ,7 , weight=4)
25 G.add_edge(1 ,8 , weight=5)
26 G.add_edge(1 ,8 , weight=4)
27 G.add_edge(1 ,8 , weight=2)
28 G.add_edge(8 ,9 , weight=5)
29 G.add_edge(8 ,9 , weight=4)
30
31 blue=[(1 ,2) ,(2 ,3) ,(1 ,4) ,(4 ,5) ,(1 ,6) ,(6 ,7) ,(1 ,8) ,(8 ,9)]
32 red=[(1 ,2) ,(2 ,3) ,(1 ,4) ,(4 ,5) ,(1 ,6) ,(6 ,7) ,(1 ,8) ,(8 ,9)]
33 green=[(1 ,2) ,(2 ,3) ,(1 ,8) ]
34
35 pos = {1:(400 , 400) , 2:(300 ,300) , 3:(100 , 100) , 4:(500 ,500) , 5:(650 ,650) ,6:(250 ,500)
36     , 7:(100
37         ,700) , 8:(500 , 300) , 9:(700 ,300)}
38
39 nx.draw_networkx_nodes(G, pos , node_size=400, node_color='y' , node_shape='o')
40
41 nx.draw_networkx_edges(G, pos , edgelist=blue , width=6, alpha=0.5,
42 edge_color='b' , style='dashed')
43 nx.draw_networkx_edges(G, pos , edgelist=green , width=5, alpha=0.5,

```

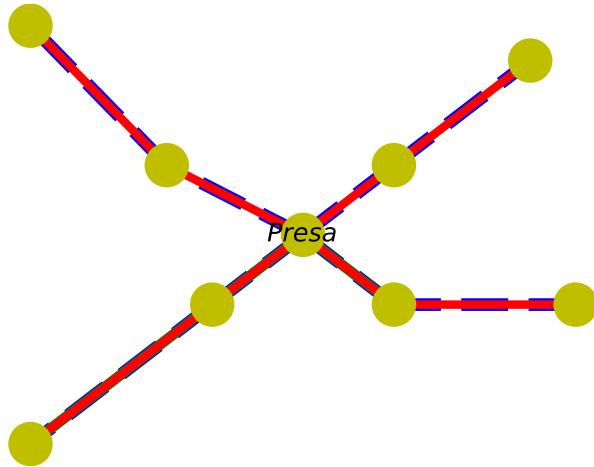


Figura 8: Canales para irrigación del suelo creados en la presa El cacao.

```

43 edge_color='g')
44 nx.draw_networkx_edges(G, pos, edgelist=red ,width=4, alpha=0.5,
45 edge_color='r')
46 labels = {}
47 labels[1] = r'$Presa$',
48
49 nx.draw_networkx_labels(G, pos, labels , font_size=12)
50
51 plot.xlim(20,800)
52 plot.axis('off')
53 plot.savefig("8.eps")
54 plot.show()

```

8.py

## 9. Multigrafo no dirigido reflexivo

En la Universidad de Oriente se quiere realizar un concurso de habilidades entre las carreras de Ingeniería en Electrónica, Ingeniería en Automática, Licenciatura en Física, Ingeniería en Informática y Licenciatura en Matemática-Cibernética. Las habilidades a evaluar serán matemática y programación.

En el grafo de la figura 9 (página 14) se muestra como está organizado el concurso. Cada vértice representa a los estudiantes de una de las carreras y las aristas, con cuales estudiantes pueden participar en cada habilidad. Todos los vértices son de color rojo porque todos los estudiantes pueden competir con estudiantes de su misma carrera.

Los vértices rojos unen a los que pueden concursar entre sí en matemáticas y los azules, los que pueden concursar en programación.

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Mon Feb 11 14:15:32 2019

```

```

4 @author: Madys
5 """
6 import matplotlib.pyplot as plot
7 import networkx as nx
8
9 G = nx . MultiGraph ()
10
11 G.add_edge(1 ,2 , weight=1)
12 G.add_edge(1 ,2 , weight=3)
13 G.add_edge(2 ,3 , weight=3)
14 G.add_edge(1 ,3 , weight=3)
15 G.add_edge(4 ,5 , weight=1)
16 G.add_edge(4 ,5 , weight=3)
17
18 blue=[(1 ,2 ),(4 ,5 )]
19 red=[(1 ,2 ),(2 ,3 ),(1 ,3 ),(4 ,5 )]
20
21 pos = {1:(200 , 100) , 2:(100 ,400) , 3:(200 , 700) , 4:(500 ,700) , 5:(650 ,400)}
22
23 nx.draw_networkx_nodes(G, pos , node_size=400, node_color='r' , node_shape='o' , alpha
   =0.6)
24 nx.draw_networkx_edges(G, pos , edgelist=blue , width=6, alpha=0.5,
edge_color='b' , style='dashed')
25
26 nx.draw_networkx_edges(G, pos , edgelist=red , width=4, alpha=0.5,
edge_color='r' )
27 labels = {}
28 labels [1] = r 'Automatica'
29 labels [2] = r 'Electrica'
30 labels [3] = r 'Fisica'
31 labels [4] = r 'Informatica'
32 labels [5] = r 'Cibernetica'
33
34 nx.draw_networkx_labels(G, pos , labels , font_size=13)
35
36 plot.xlim(20 ,800)
37 plot.axis('off')
38 plot.savefig("9.eps")
39 plot.show()

```

9.py

## 10. Multigrafo dirigido acíclico

Estos grafos pueden utilizarse para representar la cuenca hidrográfica de un río en su flujo hacia el mar. En la figura 10 (página 15) Los vértices representan los puntos en los que al menos dos ramificaciones del delta confluyen o se separan.

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Mon Feb 11 16:44:18 2019
4 @author: Madys
5 """
6
7 import matplotlib.pyplot as plot
8 import networkx as nx
9
10 G = nx . MultiDiGraph ()
11
12 G.add_edge(1 ,2 , weight=3)
13 G.add_edge(1 ,2 , weight=5)

```

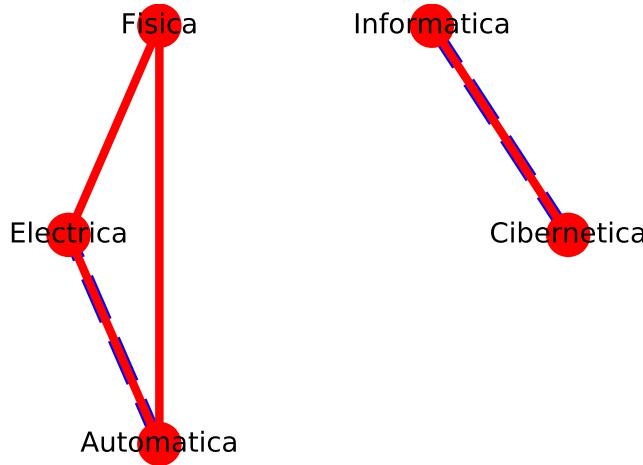


Figura 9: Concurso de habilidades.

```

14 G.add_edge(1,2, weight=6)
15 G.add_edge(2,3, weight=4)
16 G.add_edge(2,5, weight=3)
17 G.add_edge(2,5, weight=5)
18 G.add_edge(1,4, weight=4)
19 G.add_edge(1,4, weight=3)
20 G.add_edge(4,5, weight=5)
21 G.add_edge(4,5, weight=3)
22 G.add_edge(1,6, weight=3)
23 G.add_edge(1,6, weight=2)
24 G.add_edge(6,7, weight=1)
25 G.add_edge(6,7, weight=4)
26 G.add_edge(1,8, weight=5)
27 G.add_edge(1,8, weight=4)
28 G.add_edge(1,8, weight=2)
29 G.add_edge(8,9, weight=5)
30 G.add_edge(8,9, weight=4)
31
32 blue=[(1,2),(2,3),(2,5),(1,4),(4,5),(1,6),(6,7),(1,8),(8,9)]
33 red=[(1,2),(2,5),(1,4),(4,5),(1,6),(6,7),(1,8),(8,9)]
34 green=[(1,2),(2,5),(1,8)]
35
36 pos = {1:(400, 700), 2:(300,300), 3:(150, 100), 4:(400,200), 5:(300,100),6:(250,500)
37 , 7:(100
38 ,100), 8:(500, 300), 9:(700,100)}
39
40 nx.draw_networkx_nodes(G, pos, node_size=400, node_color='y', node_shape='o')
41 nx.draw_networkx_edges(G, pos, edgelist=blue, width=6, alpha=0.5,
42 edge_color='b', style='dashed')
43 nx.draw_networkx_edges(G, pos, edgelist=green, width=5, alpha=0.5,
44 edge_color='g')
45 nx.draw_networkx_edges(G, pos, edgelist=red, width=4, alpha=0.5,
46 edge_color='r')
47 labels = {}
48 labels[1] = r'Nacimiento'
49
50 nx.draw_networkx_labels(G, pos, labels, font_size=12)
51 plot.xlim(20,800)

```

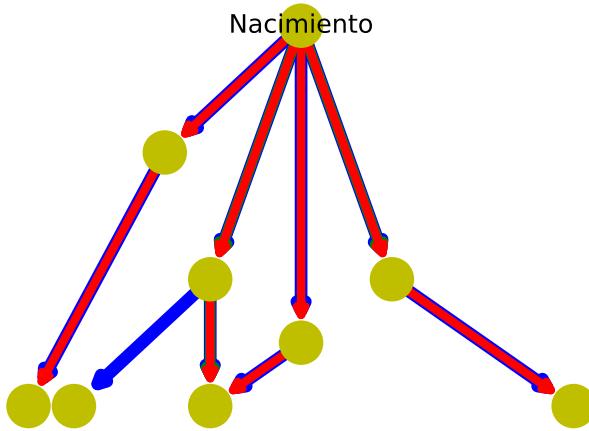


Figura 10: Representación del delta de un río en su transcurso hacia el mar.

```

52 plot.axis('off')
53 plot.savefig("10.eps")
54 plot.show()

```

10.py

## 11. Multigrafo dirigido cíclico

Este tipo de grafo es especialmente útil para representar viajes por lugares a los que se puede llegar mediante diferentes vías, regresando luego al punto de origen. En la figura 4 (página 6), se observa que un ejemplo concreto de esto sería un recorrido vacacional por varias ciudades. Partiendo de la ciudad A, hay tres posibles vías para llegar a la ciudad B, en bus, en auto de alquiler o en tren, cada uno de estos medios de transporte representaría una arista diferente para unir los vértices. Así sucesivamente, se tienen en cuenta las posibles vías de transporte hacia las distintas ciudades hasta completar el recorrido y regresar a casa.

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Mon Feb 11 17:00:30 2019
4 @author: Madys
5 """
6
7 import matplotlib.pyplot as plot
8 import networkx as nx
9
10 G = nx.MultiDiGraph()
11
12 G.add_edge(1,2, weight=3)
13 G.add_edge(1,2, weight=5)
14 G.add_edge(1,2, weight=6)
15 G.add_edge(2,3, weight=3)
16 G.add_edge(2,3, weight=5)
17 G.add_edge(2,3, weight=6)
18 G.add_edge(3,4, weight=3)

```

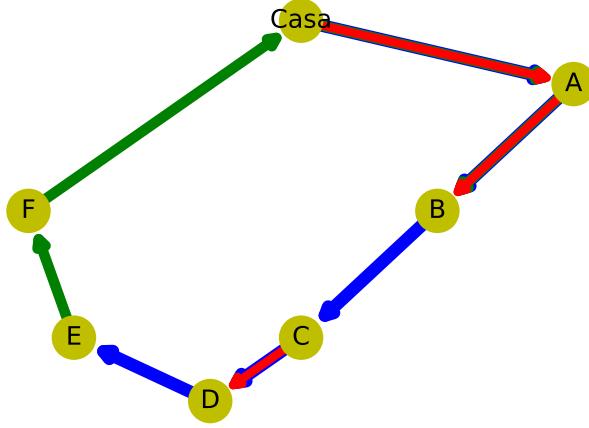


Figura 11: Representación de las posibles maneras de elegir la ruta de un circuito vacacional.

```

19 G.add_edge(4,5, weight=3)
20 G.add_edge(4,5, weight=5)
21 G.add_edge(5,6, weight=3)
22 G.add_edge(6,7, weight=6)
23 G.add_edge(7,1, weight=6)
24
25 blue=[(1,2),(2,3),(3,4),(4,5),(5,6)]
26 red=[(1,2),(2,3),(4,5)]
27 green=[(1,2),(2,3),(6,7),(7,1)]
28
29 pos = {1:(400, 700), 2:(700,600), 3:(550, 400), 4:(400,200), 5:(300,100),6:(150,200),
30 , 7:(100
31 ,400)}
32
33 nx.draw_networkx_nodes(G, pos, node_size=400, node_color='y', node_shape='o')
34 nx.draw_networkx_edges(G, pos, edgelist=blue, width=6, alpha=0.5,
35 edge_color='b', style='dashed')
36 nx.draw_networkx_edges(G, pos, edgelist=green, width=5, alpha=0.5,
37 edge_color='g')
38 nx.draw_networkx_edges(G, pos, edgelist=red, width=4, alpha=0.5,
39 edge_color='r')
40
41 labels = {}
42 labels[1] = r'Casa'
43 labels[2] = r'A'
44 labels[3] = r'B'
45 labels[4] = r'C'
46 labels[5] = r'D'
47 labels[6] = r'E'
48 labels[7] = r'F'
49
50 nx.draw_networkx_labels(G, pos, labels, font_size=12)
51 plot.xlim(20,800)
52 plot.axis('off')
53 plot.savefig("11.eps")
54 plot.show()

```

11.py

## 12. Multigrafo dirigido reflexivo

González-Cervantes [3] empleó teoría de grafos para representar el potencial eléctrico en el corazón, demostrando que se pueden incorporar las leyes fisiológicas involucradas. Cada uno de los vértices representa uno de los puntos principales que generan los impulsos eléctricos y que llevan la electricidad a cada parte del corazón, las aristas describen el valor máximo de voltaje y su duración en tiempo que descarga cada vértice. Además, puede proporcionar información con respecto al potencial eléctrico por zonas para una mejor localización. En la figura 12 (página 18) se muestra la representación con grafos del intervalo PR.

```
1 """
2 Created on Mon Feb 11 17:00:30 2019
3 @author: Madys
4 """
5 import matplotlib.pyplot as plot
6 import networkx as nx
7
8 G = nx . MultiDiGraph ()
9
10 G.add_edge(1 ,2)
11 G.add_edge(2 ,3)
12 G.add_edge(3 ,4)
13 G.add_edge(3 ,5)
14 G.add_edge(1 ,6 , weight=3)
15 G.add_edge(1 ,6 , weight=1)
16 blue=[(1 ,6)]
17 G.add_edge(2 ,6)
18 node1 = {6}
19
20 pos = {1:(50 , 350) , 2:(250 ,350) , 3:(450 , 350) , 4:(600 ,350) , 5:(550 ,340) , 6:(450 ,330)
21 }
22 nx.draw_networkx_nodes(G, pos , node_size=500, node_color='y' , node_shape='o')
23 nx.draw_networkx_edges(G, pos , width=1, alpha=0.8, edge_color='black')
24 nx.draw_networkx_nodes(G, pos , nodelist=node1 , node_size=400, node_color='r' ,
25 node_shape='o')
26 nx.draw_networkx_edges(G, pos , edgelist=blue , width=6, alpha=0.5,
27 edge_color='b' , style='dashed')
28 labels = {}
29 labels [1] = r 'v1'
30 labels [2] = r 'v2'
31 labels [3] = r 'v3'
32 labels [4] = r 'e21'
33 labels [5] = r 'e14'
34 labels [6] = r 'v6'
35 nx.draw_networkx_labels(G, pos , labels , font_size=12)
36 plot.xlim(20 ,800)
37 plot.axis('off')
38 plot.savefig("12.eps")
39 plot.show()
```

12.py

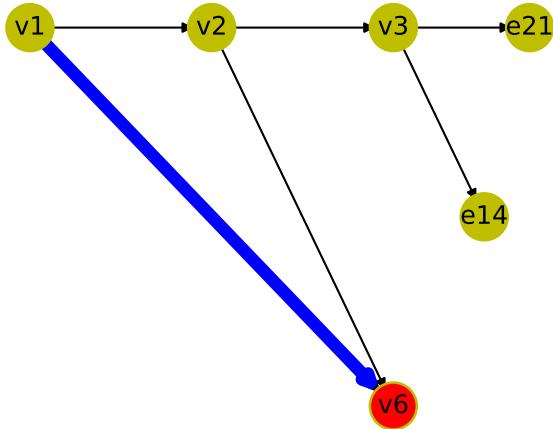
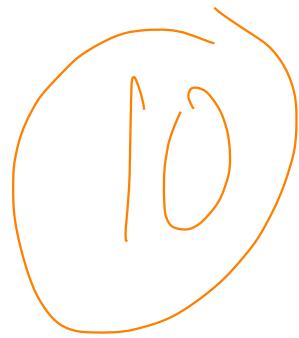


Figura 12: Representación con grafos del intervalo PR.

## Referencias

- [1] Ravindra K Ahuja. *Network flows: theory, algorithms, and applications*. Pearson Education, 2017.
- [2] Fan Chung. Graph theory in the information age. *Notices of the AMS*, 57(6):726–732, 2010.
- [3] Natalia González-Cervantes, Aurora Espinoza-Valdez, and Ricardo Salido-Ruiz. Potencial eléctrico en el corazón: Representación mediante un grafo. *ReCIBE*, 5(3), 2016.
- [4] Amador Menéndez Velázquez. Una breve introducción a la teoría de grafos. *Suma*, 28:11–26, 1998.

Tarea 2  
5280  
25 de febrero de 2019



## Introducción

La teoría de grafos tiene aplicación en diversas áreas, debido a que las redes aparecen prácticamente en todas las actividades de nuestra vida diaria: sistemas de comunicaciones, sistemas hidráulicos, circuitos eléctricos y electrónicos, sistemas mecánicos, redes sociales, entre otros.

Como constata Ravindra [1], las redes físicas son quizás las más comunes y mejor identificables. Los sistemas de transporte, cualquiera que sea el medio suelen modelarse en sistemas de distribución y decisiones logísticas complejos, siendo el **Problema de transporte** un ejemplo típico de investigación de operaciones.

En este problema, un transportista con inventario de mercancías en sus almacenes debe enviar estos productos a centros minoristas geográficamente dispersos, cada uno con una demanda dada del cliente, donde el objetivo es incurrir en los mínimos gastos de transporte posibles. Una de las maneras de resolver este problemas es mediante algoritmos computacionales basados en grafos.

Otro ejemplo representativo de uso de grafos son los algoritmos de búsqueda web de Google. Estos se basan en el gráfico WWW, que contiene todas las páginas web como vértices y los hipervínculos como bordes [2].

La teoría de los grafos también ~~ha sido~~ <sup>es</sup> usada en química, debido a la posibilidad de representar los modelos estructurales mediante diagramas. En un grafo molecular, los vértices representan a los átomos y los lados a los enlaces químicos que conectan ciertas parejas de átomos [6].

### 1. Grafo simple no dirigido acíclico

La mayoría de los esquemas de líneas de autobuses, tranvías o trenes del transporte público pueden ser representados por grafos simples no dirigidos acíclicos.

En la figura 1 se muestra una sección de ocho estaciones de la Línea 2 del Metro de Monterrey, donde las estaciones son los nodos y el tramo de línea entre ellos, los vértices.

Para posicionar los nodos de este grafo se empleó el **spring layout**. Se obtuvieron resultados igualmente satisfactorios empleando otros **layouts**.



Figura 1: Representación con un grafo de ocho estaciones de la Línea 2 del Metro de Monterrey.

```

1 import matplotlib.pyplot as plot
2 import networkx as nx
3
4 G = nx . Graph ()
5 G.add_edge('Sendero','Santiago Tapia')
6 G.add_edge('Santiago Tapia','San Nicolas')
7 G.add_edge('San Nicolas','Anahuac')
8 G.add_edge('Anahuac','Universidad')
9 G.add_edge('Universidad','Ninos Heroes')
10 G.add_edge('Ninos Heroes','Regina')
11 G.add_edge('Regina','General Anaya')
12
13 positions = nx.spring_layout(G, scale=.15)
14 nx.draw_networkx_nodes(G, positions, node_size=500, node_color='y')
15 nx.draw_networkx_edges(G, positions, width=1, alpha=1, edge_color='b')
16 nx.draw_networkx_labels(G, positions, font_size=12, font_family='sans-serif')
17 plot.axis('off')
18 plot.savefig("fig1.eps")
19 plot.show()

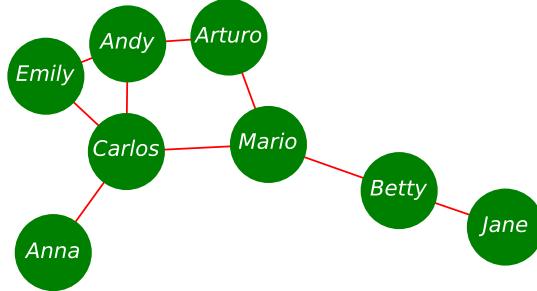
```

Listing 1: Representación con un grafo de ocho estaciones de la Línea 2 del Metro de Monterrey.

## 2. Grafo simple no dirigido cíclico

Este tipo de grafos es adecuado para representar relaciones comerciales entre empresas al igual que relaciones de parentezco, amistad o romance entre personas. En la figura 2 se muestra un grafo que representa las relaciones de amistad en un grupo de diez personas. Los vértices son las personas y las aristas, las personas que tienen una relación de amistad.

Para posicionar este grafo se usó el Fruchterman-Reingold layout. Este fue diseñado para dibujar gráficos no



\vspace{8mm}

Figura 2: Relaciones de amistad en un grupo de personas.

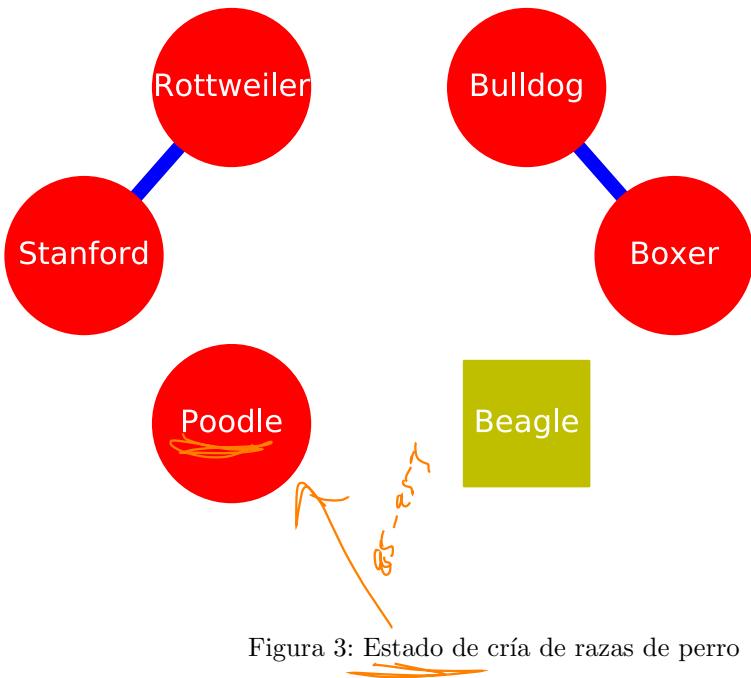
dirigidos, con el objetivo de distribuir los vértices uniformemente en el marco, haciendo que las longitudes de los bordes sean uniformes y reflejando la simetría [3].

```

1 import matplotlib.pyplot as plot
2 import networkx as nx
3 G = nx . Graph ()
4 G.add_edge(1 ,2)
5 G.add_edge(1 ,3)
6 G.add_edge(1 ,4)
7 G.add_edge(4 ,5)
8 G.add_edge(2 ,6)
9 G.add_edge(3 ,8)
10 G.add_edge(7 ,8)
11 G.add_edge(4 ,8)
12 G.add_edge(4 ,7)
13 pos=nx . fruchterman_reingold_layout(G, scale=.45)
14 nx . draw_networkx_nodes(G, pos, node_size=1800, node_color='g', node_shape='o')
15 nx . draw_networkx_edges(G, pos, width=1, alpha=0.5, edge_color='red')
16 labels = {}
17 labels[1] = r '$ Mario $'
18 labels[2] = r '$Betty$'
19 labels[3] = r '$Arturo$'
20 labels[4] = r '$Carlos$'
21 labels[5] = r '$Anna$'
22 labels[6] = r '$Jane$'
23 labels[7] = r '$Emily$'
24 labels[8] = r '$Andy$'
25 nx . draw_networkx_labels(G, pos, labels, font_size=12, font_color='w')
26 plot . axis('off')
27 plot . savefig("fig2 . eps")
28 plot . show()

```

Listing 2: Relaciones de amistad en un grupo de personas.



### 3. Grafo simple no dirigido reflexivo

Un criador de perros posee seis razas diferentes de estos animales. Con un grafo de este tipo puede representar de cuales de estos animales ha obtenido crías, siendo los vértices las razas de perro y las aristas la unión entre razas que han tenido crías.

En la figura 3 se puede observar que todas las razas excepto la Beagle, que es recién adquirida y aún no se ha reproducido, han tenido crías con su misma raza (vértices en rojo). También se observa que han habido cruzamientos entre bóxer y bulldog y entre pitbull y rottweiler.

Para posicionar este grafo se usó el Kamada-Kawai layout, diseñado para dibujar grafos ponderados no dirigidos [5]. Los layouts circular y shell mostraron también resultados similares.

```

1 import matplotlib.pyplot as plot
2 import networkx as nx
3
4 G = nx . Graph ()
5
6 G.add_edge(1 ,2)
7 G.add_edge(3 ,4)
8 G.add_node(5)
9 G.add_node(6)
10 apareados={1 ,2 ,3 ,4 ,5}
11 noApareado={6}
12 pos = nx.kamada_kawai_layout(G, scale=.45)
13
14 nx.draw_networkx_nodes(G, pos ,nodelist=apareados ,node_size=4400, node_color='r' , node_shape='o')
15 nx.draw_networkx_nodes(G, pos ,nodelist=noApareado , node_size=2800, node_color='y' , node_shape='s')
16 nx.draw_networkx_edges(G, pos ,width=6, alpha=0.6, edge_color='b')
17

```

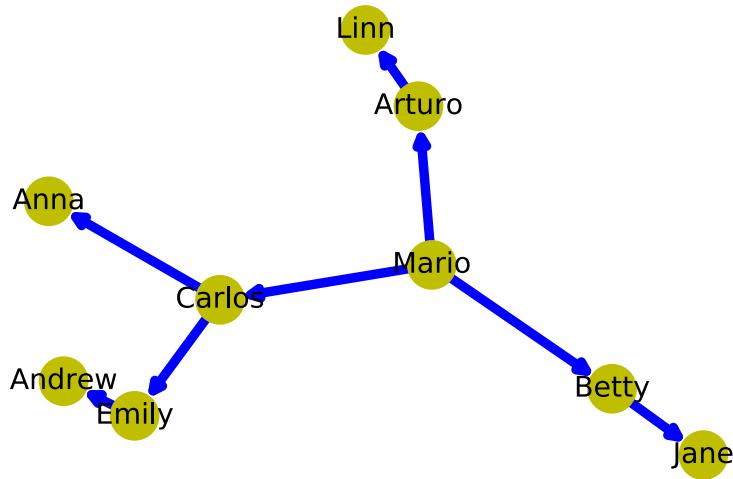


Figura 4: Representación de la transmisión de una ITS en un grupo de personas.

```

18 labels = []
19 labels[1] = r'Boxer'
20 labels[2] = r'Bulldog'
21 labels[3] = r'Rottweiler'
22 labels[4] = r'Stanford'
23 labels[5] = r'Poodle'
24 labels[6] = r'Beagle'
25
26 nx.draw_networkx_labels(G, pos, labels, font_size=13, font_color='w')
27
28 plot.axis('off')
29 plot.savefig("fig3.eps")
30 plot.show()

```

Listing 3: Estado de cría de razas de perro.

## 4. Grafo simple dirigido acíclico

La cadena de propagación de las ITS (Infecciones de Transmisión Sexual) para las cuales no se conoce cura o las que solo afectan al individuo una vez en la vida, pueden representarse mediante grafos simples dirigidos acíclicos. En este caso cada sujeto es un vértice y la dirección de transmisión de la enfermedad es representada en la arista ~~Mari~~. Desde la enfermedad se propaga desde la persona portadora hacia el sujeto sano que posteriormente se convierte en portador y contagia a otros sujetos sanos.

De los layouts empleados para posicionar los elementos de la figura 4 el Fruchterman-Reingold fue el más adecuado.

```

1 import matplotlib.pyplot as plot
2 import networkx as nx
3
4 G = nx . DiGraph ()
5 G.add_edge(1 ,2)
6 G.add_edge(1 ,3)
7 G.add_edge(1 ,4)
8 G.add_edge(4 ,5)
9 G.add_edge(2 ,6)
10 G.add_edge(7 ,8)
11 G.add_edge(4 ,7)
12 G.add_edge(3 ,9)
13 pos = nx . fruchterman_reingold_layout (G, scale=.55, iterations=30)
14 nx.draw_networkx_nodes(G, pos, node_size=400, node_color='y', node_shape='o')
15 nx.draw_networkx_edges(G, pos, width=4, edge_color='b')
16
17 labels = {}
18 labels[1] = r'Mario'
19 labels[2] = r'Betty'
20 labels[3] = r'Arturo'
21 labels[4] = r'Carlos'
22 labels[5] = r'Anna'
23 labels[6] = r'Jane'
24 labels[7] = r'Emily'
25 labels[8] = r'Andrew'
26 labels[9] = r'Linn'
27
28 nx.draw_networkx_labels(G, pos, labels, font_size=12)
29 plot.axis('off')
30 plot.savefig("fig4 .eps")
31 plot.show()

```

Listing 4: Representación de la transmisión de una ITS en un grupo de personas.

## 5. Grafo simple dirigido cíclico

En ciudades con carreteras estrechas como Santiago de Cuba, en diferentes sectores, las carreteras son de un solo sentido y se representan con grafos de este tipo. Las aristas representan las calles y los vértices son las intersecciones entre al menos dos aristas.

Luego de probar varios layouts, el spectral fue el que arrojó el resultado deseado para dibujar la figura 5.

```

1 import matplotlib.pyplot as plot
2 import networkx as nx
3
4 G = nx . DiGraph ()
5
6 G.add_edge(1 ,2, calle='c')
7 G.add_edge(2 ,3, calle='c')
8 G.add_edge(5 ,4, calle='b')
9 G.add_edge(6 ,5, calle='b')
10 G.add_edge(8 ,9, calle='a')
11 G.add_edge(7 ,8, calle='a')
12 G.add_edge(1 ,4, calle='19')
13 G.add_edge(4 ,7, calle='19')
14 G.add_edge(8 ,5, calle='21')
15 G.add_edge(5 ,2, calle ='21')
16 G.add_edge(3 ,6, calle ='23')
17 G.add_edge(6 ,9, calle ='23')
18

```

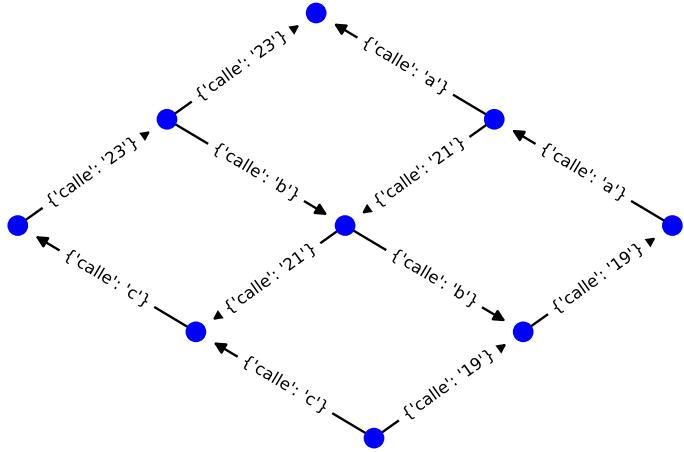


Figura 5: Carreteras de sentido único en Santiago de Cuba.

```

19 pos = nx.spectral_layout(G)
20 nx.draw_networkx_nodes(G, pos, node_size=60, node_color='b', node_shape='o')
21 nx.draw_networkx_edges(G, pos, width=1, edge_color='black')
22 edge_labels=nx.draw_networkx_edge_labels(G,pos,font_size=7,label_pos=.5)
23
24
25 plot.axis('off')
26 plot.savefig("fig5.eps")
27 plot.show()

```

Listing 5: Representación de carreteras de doble sentido en Santiago de Cuba.

## 6. Grafo simple dirigido reflexivo

Un grafo en el que cada vértice es una empresa de servicios y las aristas, la unión con cada una de las otras empresas a las que le presta servicios, puede representarse mediante un grafo dirigido reflexivo, pues algunas de estas empresas se brindan servicio a ellas mismas.

Un sitio web pequeño, en el cual se accede a todas sus páginas a través de un menú estático también se puede representar con este tipo de grafo.

Otro ejemplo está dado por la representación de un grupo de personas conectadas a una red social y los perfiles de otras personas que tengan abiertos en el navegador en ese momento. Cada una de esas persona también pudiesen estar mirando su propio perfil. En el grafo de la figura 6 los vértices son las personas y las aristas indican el perfil de qué personas tiene cada uno abierto en el navegador. Puede apreciarse en color rojo que Mayra y Andrew tienen abiertos sus propios perfiles.

Para obtener la forma deseada del grafo dieron resultados similares el circular layout y el shell.

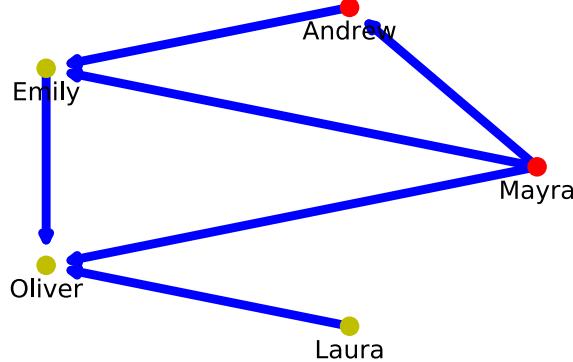


Figura 6: Representación de personas mirando perfiles en redes sociales.

```

1 import matplotlib.pyplot as plot
2 import networkx as nx
3
4 G = nx . DiGraph ()
5 G.add_edge(1,2)
6 G.add_edge(1,3)
7 G.add_edge(1,5)
8 G.add_edge(2,3)
9 G.add_edge(3,5)
10 G.add_edge(4,5)
11 node1 = {1,2}
12 node2 = {3,4,5}
13 pos = nx.circular_layout(G)
14 nx.draw_networkx_nodes(G, pos, nodelist=node1, node_size=100, node_color='r', node_shape='o',
15 alpha=0.8)
15 nx.draw_networkx_nodes(G, pos, nodelist=node2, node_size=100, node_color='y', node_shape='o')
16 nx.draw_networkx_edges(G, pos, width=5, alpha=0.4, edge_color='b')
17
18 labels = {}
19 labels[1] = r'Mayra'
20 labels[2] = r'Andrew'
21 labels[3] = r'Emily'
22 labels[4] = r'Laura'
23 labels[5] = r'Oliver',
24
25 for i in pos:
26     pos[i][1] = pos[i][1] - 0.15
27 nx.draw_networkx_labels(G, pos, labels, font_size=14)
28
29 plot.axis('off')
30 plot.savefig("fig6.eps")
31 plot.show()

```

Listing 6: Representación de personas mirando perfiles en redes sociales.

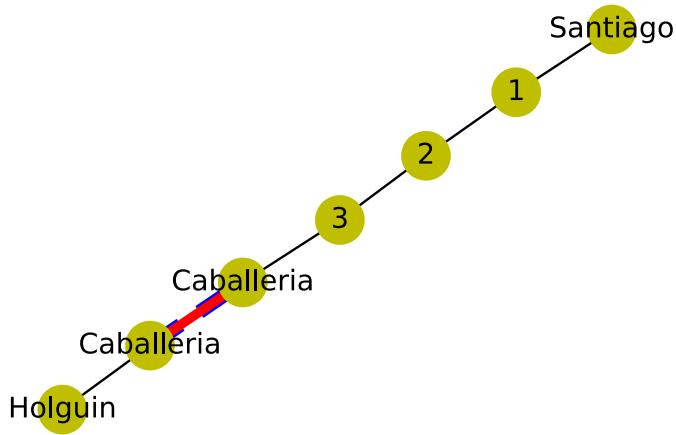


Figura 7: Posible ruta entre Santiago de Cuba y Holguín

## 7. Multigrafo no dirigido acíclico

La ruta entre Santiago de Cuba y Holguín puede representarse como un multigrafo no dirigido cíclico. En un pueblo llamado Caballería la carretera principal se bifurca, pudiendo continuar por la ruta principal hasta Banes, o por el camino alternativo de Caballería, que, aunque es más largo, puede recorrerse en menor tiempo a causa del poco tránsito.

En la figura 7 los vértices representan las cabeceras municipales, que están simplemente enumeradas por no ser de interés para el ejemplo mencionado. Para la visualización se empleó el layout Kamada-Kawai.

```

1 import matplotlib.pyplot as plot
2 import networkx as nx
3
4 G = nx . MultiGraph ()
5
6
7 G.add_edge(1,2, weight=1)
8 G.add_edge(2,3 ,weight=1)
9 G.add_edge(3,4, weight=1)
10 G.add_edge(4,5, weight=1)
11 G.add_edge(5,6, weight=1)
12 G.add_edge(5,6,weight=2)
13 G.add_edge(6,7, weight=1)
14 black=[(1,2),(2,3),(3,4),(4,5),(6,7)]
15 blue=[(5,6)]
16 red=[(5,6)]
17 pos = nx.kamada_kawai_layout(G, scale=.2)
18
19 nx.draw_networkx_nodes(G, pos ,node_size=400, node_color='y', node_shape='o')
20 nx.draw_networkx_edges(G, pos, edgelist=black, width=1,edge_color='black', alpha=0.8)
21 nx.draw_networkx_edges(G, pos, edgelist=blue ,width=6, alpha=0.5,
22 edge_color='b' , style='dashed')
```

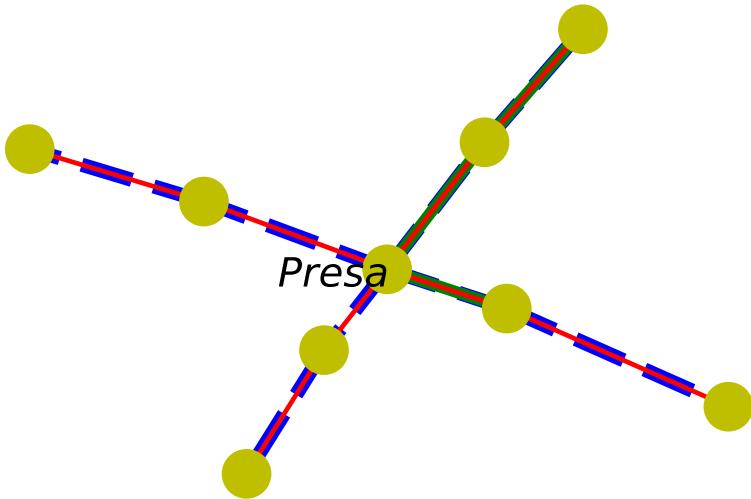


Figura 8: Canales para irrigación del suelo creados en la presa El cacao.

```

23 nx.draw_networkx_edges(G, pos, edgelist=red, width=4, alpha=0.5,
24 edge_color='r')
25 labels = {}
26 labels[1] = r'Santiago'
27 labels[2] = r'1'
28 labels[3] = r'2'
29 labels[4] = r'3'
30 labels[5] = r'Caballeria'
31 labels[6] = r'Caballeria'
32 labels[7] = r'Holguin'
33
34 nx.draw_networkx_labels(G, pos, labels, font_size=12)
35
36 plot.axis('off')
37 plot.savefig("fig7.eps")
38 plot.show()

```

Listing 7: Posible ruta entre Santiago de Cuba y Holguín.

## 8. Multigrafo no dirigido cíclico

En la presa El cacao, ubicada en el Municipio Cotorro, Ciudad de La Habana, se realizó un proyecto para favorecer la agricultura. Al rededor de la presa se construyeron una serie de canales para mantener el suelo húmedo durante todo el año. Este sistema puede ser representado mediante el grafo de la figura 8, el cual los vértices representan la unión entre uno o más canales y las aristas, cada uno de los canales. El layout Kamada-Kawai fue el que mejores resultados de visualización arrojó.

```

1 import matplotlib.pyplot as plot
2 import networkx as nx
3
4 G = nx . MultiGraph ()
5
6 G.add_edge(1,2, weight=3)
7 G.add_edge(1,2, weight=5)
8 G.add_edge(1,2, weight=6)
9 G.add_edge(2,3, weight=4)
10 G.add_edge(2,3, weight=3)
11 G.add_edge(2,3, weight=5)
12 G.add_edge(1,4, weight=4)
13 G.add_edge(1,4, weight=3)
14 G.add_edge(4,5, weight=5)
15 G.add_edge(4,5, weight=3)
16 G.add_edge(1,6, weight=3)
17 G.add_edge(1,6, weight=2)
18 G.add_edge(6,7, weight=3)
19 G.add_edge(6,7, weight=4)
20 G.add_edge(1,8, weight=5)
21 G.add_edge(1,8, weight=4)
22 G.add_edge(1,8, weight=2)
23 G.add_edge(8,9, weight=5)
24 G.add_edge(8,9, weight=4)
25
26 blue=[(1,2),(2,3),(1,4),(4,5),(1,6),(6,7),(1,8),(8,9)]
27 red=[(1,2),(2,3),(1,4),(4,5),(1,6),(6,7),(1,8),(8,9)]
28 green=[(1,2),(2,3),(1,8)]
29
30 pos = nx.kamada_kawai_layout(G)
31
32 nx.draw_networkx_nodes(G, pos ,node_size=400, node_color='y', node_shape='o')
33 nx.draw_networkx_edges(G, pos , edgelist=blue, width=6, alpha=0.5,
34 edge_color='b', style='dashed')
35 nx.draw_networkx_edges(G, pos , edgelist=green, width=5, alpha=0.5,
36 edge_color='g')
37 nx.draw_networkx_edges(G, pos , edgelist=red, width=2, alpha=0.5,
38 edge_color='r')
39 labels = {}
40 labels [1] = r'$Presa$'
41 for i in pos:
42     pos[i][0] = pos[i][0] -0.15
43     pos[i][1] = pos[i][1] -0.03
44 nx.draw_networkx_labels(G, pos , labels , font_size=17)
45
46 plot.axis('off')
47 plot.savefig("fig8.eps")
48 plot.show()

```

Listing 8: Canales para irrigación del suelo creados en la presa El cacao.

## 9. Multigrafo no dirigido reflexivo

En la Universidad de Oriente se quiere realizar un concurso de habilidades entre las carreras de Ingeniería en Electrónica, Ingeniería en Automática, Licenciatura en Física, Ingeniería en Informática y Licenciatura en Matemática-Cibernética. Las habilidades a evaluar serán matemática y programación.

La figura 9 muestra como está organizado el concurso. Cada vértice representa a los estudiantes que estudian una de las carreras y las aristas, con qué estudiantes pueden participar en cada habilidad. Todos los vértices son de color rojo porque todos los estudiantes pueden competir con estudiantes de su misma carrera.

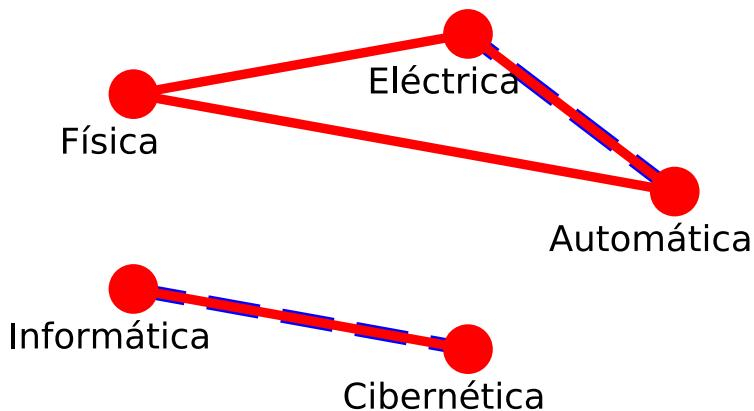


Figura 9: Concurso de habilidades.

Los vértices rojos unen a los que pueden concursar entre sí en matemáticas y los azules, los que pueden concursar en programación.

Para la visualización se empleó el layout circular.

```

1 import matplotlib.pyplot as plot
2 import networkx as nx
3
4 G = nx . MultiGraph ()
5
6 G.add_edge(1 ,2 , weight=1)
7 G.add_edge(1 ,2 , weight=3)
8 G.add_edge(2 ,3 , weight=3)
9 G.add_edge(1 ,3 , weight=3)
10 G.add_edge(4 ,5 , weight=1)
11 G.add_edge(4 ,5 , weight=3)
12
13 blue=[(1 ,2) ,(4 ,5)]
14 red=[(1 ,2) ,(2 ,3) ,(1 ,3) ,(4 ,5)]
15
16 pos = nx.circular_layout(G, scale=0.1)
17
18 nx.draw_networkx_nodes(G, pos ,node_size=400, node_color='r' , node_shape='o')
19 nx.draw_networkx_edges(G, pos , edgelist=blue ,width=6, alpha=0.5,
20 edge_color='b' , style='dashed')
21
22 nx.draw_networkx_edges(G, pos , edgelist=red ,width=4, alpha=0.5,
23 edge_color='r')
24 labels = {}
25 labels [1] = r'Automatica'
26 labels [2] = r'Electrica'
27 labels [3] = r'Fisica'
28 labels [4] = r'Informatica'
29 labels [5] = r'Cibernetica'

```

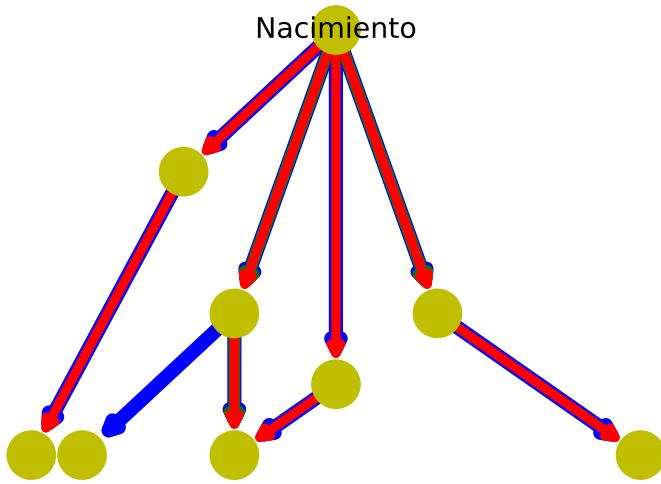


Figura 10: Representación del delta de un río en su transcurso hacia el mar.

```

30
31 for i in pos:
32     pos[i][1] = pos[i][1] - 0.03
33     pos[i][0] = pos[i][0] - 0.008
34 nx.draw_networkx_labels(G, pos, labels, font_size=15)
35
36 plot.axis('off')
37 plot.savefig("fig9.eps")
38 plot.show()

```

Listing 9: Concurso de habilidades.

## 10. Multigrafo dirigido acíclico

Estos grafos pueden utilizarse para representar la cuenca hidrográfica de un río en su flujo hacia el mar. En la figura 10 los vértices representan los puntos en los que al menos dos ramificaciones del delta confluyen o se separan. Para que el grafo tuviese exactamente la forma deseada, se insertaron las coordenadas de los nodos de manera manual.

```

1 import matplotlib.pyplot as plot
2 import networkx as nx
3
4 G = nx.MultiDiGraph()
5
6 G.add_edge(1,2, weight=3)
7 G.add_edge(1,2, weight=5)
8 G.add_edge(1,2, weight=6)
9 G.add_edge(2,3, weight=4)
10 G.add_edge(2,5, weight=3)

```

```

11 G.add_edge(2,5, weight=5)
12 G.add_edge(1,4, weight=4)
13 G.add_edge(1,4, weight=3)
14 G.add_edge(4,5, weight=5)
15 G.add_edge(4,5, weight=3)
16 G.add_edge(1,6, weight=3)
17 G.add_edge(1,6, weight=2)
18 G.add_edge(6,7, weight=1)
19 G.add_edge(6,7, weight=4)
20 G.add_edge(1,8, weight=5)
21 G.add_edge(1,8, weight=4)
22 G.add_edge(1,8, weight=2)
23 G.add_edge(8,9, weight=5)
24 G.add_edge(8,9, weight=4)
25
26 blue=[(1,2),(2,3),(2,5),(1,4),(4,5),(1,6),(6,7),(1,8),(8,9)]
27 red=[(1,2),(2,5),(1,4),(4,5),(1,6),(6,7),(1,8),(8,9)]
28 green=[(1,2),(2,5),(1,8)]
29
30 pos = {1:(400, 700), 2:(300,300), 3:(150, 100), 4:(400,200), 5:(300,100),6:(250,500), 7:(100
31 ,100), 8:(500, 300), 9:(700,100)}
32 nx.draw_networkx_nodes(G, pos, node_size=400, node_color='y', node_shape='o')
33 nx.draw_networkx_edges(G, pos, edgelist=blue, width=6, alpha=0.5,
34 edge_color='b', style='dashed')
35 nx.draw_networkx_edges(G, pos, edgelist=green, width=5, alpha=0.5,
36 edge_color='g')
37 nx.draw_networkx_edges(G, pos, edgelist=red, width=4, alpha=0.5,
38 edge_color='r')
39 labels = {}
40 labels[1] = r'Nacimiento'
41
42 nx.draw_networkx_labels(G, pos, labels, font_size=12)
43
44 plot.xlim(20,800)
45 plot.axis('off')
46 plot.savefig("fig10.eps")
47 plot.show()

```

Listing 10: Representación del delta de un río en su transcurso hacia el mar.

## 11. Multigrafo dirigido cíclico

Este tipo de grafo es especialmente útil para representar viajes por lugares a los que se puede llegar mediante diferentes vías, regresando luego al punto de origen. Un ejemplo concreto de esto sería un recorrido vacacional por varias ciudades. Partiendo de la ciudad A, hay tres posibles vías para llegar a la ciudad B, en bus, en auto de alquiler o en tren, cada uno de estos medios de transporte representaría una arista diferente para unir los vértices. Así sucesivamente, se tienen en cuenta las posibles vías de transporte hacia las distintas ciudades hasta completar el recorrido y regresar a casa.

Para posicionar los vértices en la figura 11 se empleó el layout shell, aunque el circular también ofrece buenos resultados en este tipo de grafos.

```

1 import matplotlib.pyplot as plot
2 import networkx as nx
3 G = nx.MultiDiGraph()
4 G.add_edge(1,2, weight=3)
5 G.add_edge(1,2, weight=5)
6 G.add_edge(1,2, weight=6)
7 G.add_edge(2,3, weight=3)
8 G.add_edge(2,3, weight=5)

```

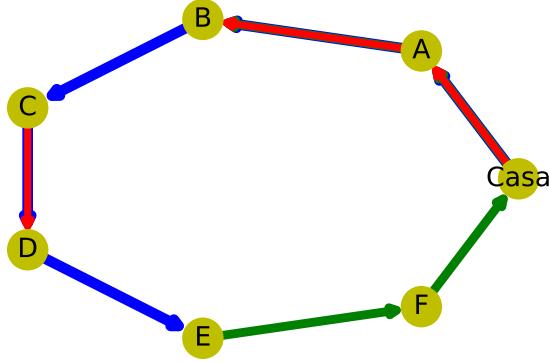


Figura 11: Representación de las posibles maneras de elegir la ruta de un circuito vacacional.

```

9 G.add_edge(2,3, weight=6)
10 G.add_edge(3,4, weight=3)
11 G.add_edge(4,5, weight=3)
12 G.add_edge(4,5, weight=5)
13 G.add_edge(5,6, weight=3)
14 G.add_edge(6,7, weight=6)
15 G.add_edge(7,1, weight=6)
16 blue=[(1,2),(2,3),(3,4),(4,5),(5,6)]
17 red=[(1,2),(2,3),(4,5)]
18 green=[(1,2),(2,3),(6,7),(7,1)]
19 pos = nx.shell_layout(G)
20 nx.draw_networkx_nodes(G, pos, node_size=500, node_color='y', node_shape='o')
21 nx.draw_networkx_edges(G, pos, edgelist=blue, width=6, alpha=0.5,
22 edge_color='b', style='dashed')
23 nx.draw_networkx_edges(G, pos, edgelist=green, width=5, alpha=.7,
24 edge_color='g')
25 nx.draw_networkx_edges(G, pos, edgelist=red, width=4, alpha=0.5,
26 edge_color='r')
27 labels = {}
28 labels[1] = r'Casa'
29 labels[2] = r'A'
30 labels[3] = r'B'
31 labels[4] = r'C'
32 labels[5] = r'D'
33 labels[6] = r'E'
34 labels[7] = r'F'
35 nx.draw_networkx_labels(G, pos, labels, font_size=15, )
36 plot.axis('off')
37 plot.savefig("fig11.eps")

```

Listing 11: Representación de las posibles maneras de elegir la ruta de un circuito vacacional.

## 12. Multigrafo dirigido reflexivo

*a*

González-Cervantes [4] empleó teoría de grafos para representar el potencial eléctrico en el corazón, demostrando que se pueden incorporar las leyes fisiológicas involucradas. Cada uno de los vértices representa uno de los

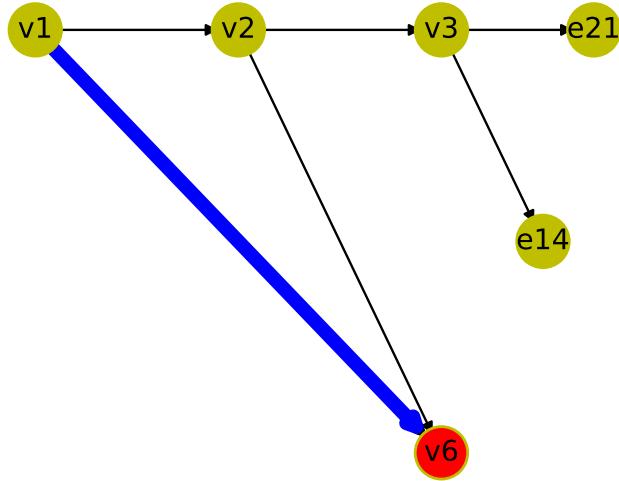


Figura 12: Representación con grafos del intervalo PR.

puntos principales que generan los impulsos eléctricos y que llevan la electricidad a cada parte del corazón, las aristas describen el valor máximo de voltaje y su duración en tiempo que descarga cada vértice. Además, puede proporcionar información con respecto al potencial eléctrico por zonas para una mejor localización. En la figura 12 se muestra la representación con grafos del intervalo PR. Debido a que se siguió la forma del corazón para dibujar el grafo, las coordenadas de los vértices fueron ingresadas manualmente.

```

1 import matplotlib.pyplot as plot
2 import networkx as nx
3
4 G = nx . MultiDiGraph ()
5
6 G.add_edge(1 ,2)
7 G.add_edge(2 ,3)
8 G.add_edge(3 ,4)
9 G.add_edge(3 ,5)
10 G.add_edge(1 ,6 ,weight=3)
11 G.add_edge(1 ,6 ,weight=1)
12 blue=[(1 ,6)]
13 G.add_edge(2 ,6)
14 node1 = {6}
15
16 pos = {1:(50 , 350) , 2:(250 ,350) , 3:(450 , 350) , 4:(600 ,350) , 5:(550 ,340) , 6:(450 ,330)}
17 nx.draw_networkx_nodes(G, pos, node_size=500, node_color='y', node_shape='o')
18 nx.draw_networkx_edges(G, pos, width=1, alpha=0.8, edge_color='black')
19 nx.draw_networkx_nodes(G, pos, nodelist=node1 ,node_size=400, node_color='r', node_shape='o')
20 nx.draw_networkx_edges(G, pos, edgelist=blue ,width=6, alpha=0.5,
21 edge_color='b' , style='dashed')
22
23 labels = {}
24 labels [1] = r 'v1'
25 labels [2] = r 'v2'
26 labels [3] = r 'v3'
27 labels [4] = r 'e21'
```

```
28 | labels[5] = r'e14'
29 | labels[6] = r'v6'
30 |
31 nx.draw_networkx_labels(G, pos, labels, font_size=12)
32 plot.xlim(20,800)
33 plot.axis('off')
34 plot.savefig("fig12.eps")
35 plot.show()
```

Listing 12: Representación con grafos del intervalo PR.

## Referencias

- [1] Ravindra K Ahuja. *Network flows: theory, algorithms, and applications*. Pearson Education, 2017.
- [2] Fan Chung. Graph theory in the information age. *Notices of the AMS*, 57(6):726–732, 2010.
- [3] Edward M Fruchterman, Thomas MJ Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.
- [4] Aurora González-Cervantes, Natalia Espinoza-Valdez and Ricardo Salido-Ruiz. Potencial eléctrico en el corazón: Representación mediante un grafo. *ReCIBE*, 5(3), 2016.
- [5] Tomihisa Kamada and Satoru others Kawai. An algorithm for drawing general undirected graphs. *Information processing letters*, 31(1):7–15, 1989.
- [6] Amador Menéndez Velázquez. Una breve introducción a la teoría de grafos. *Suma*, 28:11–26, 1998.

# Tarea 2

5280

1 de junio de 2019

## Introducción

La teoría de grafos tiene aplicación en diversas áreas, debido a que las redes aparecen prácticamente en todas las actividades de nuestra vida diaria: sistemas de comunicaciones, sistemas hidráulicos, circuitos eléctricos y electrónicos, sistemas mecánicos, redes sociales, entre otros.

Como constata Ravindra [2], las redes físicas son quizás las más comunes y mejor identificables. Los sistemas de transporte, cualquiera que sea el medio suelen modelarse en sistemas de distribución y decisiones logísticas complejos, siendo el problema de transporte un ejemplo típico de investigación de operaciones.

Otro ejemplo representativo de uso de grafos son los algoritmos de búsqueda web de Google. Estos se basan en el gráfico WWW, que contiene todas las páginas web como vértices y los hipervínculos como bordes [3].

La teoría de los grafos también es usada en química, debido a la posibilidad de representar los modelos estructurales mediante diagramas. En un grafo molecular, los vértices representan a los átomos y los lados a los enlaces químicos que conectan ciertas parejas de átomos [6].

Para acomodar los grafos, de manera que su forma refleje lo que se desea, se han desarrollado varios *layouts* : algoritmos que devuelven una lista de posiciones para los nodos, según parámetros definidos.

Entre los *layouts* más conocidos se destacan:

- *Bipartite layout*
- *Circular layout*
- *Kamada kawai layout*
- *Random layout*
- *Rescale layout*
- *Shell layout*
- *Spring layout*
- *Spectral layout*



Figura 1: Representación con un grafo de ocho estaciones de la Línea 2 del Metro de Monterrey.

## 1. Grafo simple no dirigido acíclico

La mayoría de los esquemas de líneas de autobuses, tranvías o trenes del transporte público pueden ser representados por grafos simples no dirigidos acíclicos.

En la figura 1 (página 2) se muestra una sección de ocho estaciones de la Línea 2 del Metro de Monterrey, donde las estaciones son los nodos y el tramo de línea entre ellos, los vértices.

Para posicionar los nodos de este grafo se empleó el *spring layout*. Se obtuvieron resultados igualmente satisfactorios empleando otros *layouts*.

```

1 import matplotlib.pyplot as plot
2 import networkx as nx
3
4 G = nx . Graph ()
5 G.add_edge( 'Sendero' , 'Santiago Tapia' )
6 G.add_edge( 'Santiago Tapia' , 'San Nicolas' )
7 G.add_edge( 'San Nicolas' , 'Anahuac' )
8 G.add_edge( 'Anahuac' , 'Universidad' )
9 G.add_edge( 'Universidad' , 'Ninos Heroes' )
10 G.add_edge( 'Ninos Heroes' , 'Regina' )
11 G.add_edge( 'Regina' , 'General Anaya' )
12
13 positions = nx.spring_layout(G, scale=.15)
14
15 nx.draw_networkx_nodes(G, positions , node_size=500, node_color='y')
16 nx.draw_networkx_edges(G, positions , width=1, alpha=1, edge_color='b')
17 nx.draw_networkx_labels(G, positions , font_size=12, font_family='sans-serif')
18 plot . axis( 'off' )
19 plot . savefig ("fig1 . eps")
20 plot . show()

```

Ej1.py

## 2. Grafo simple no dirigido cíclico

Este tipo de grafos es adecuado para representar relaciones comerciales entre empresas al igual que relaciones de parentesco, amistad o romance entre personas. En la figura 2 (página 3) se muestra

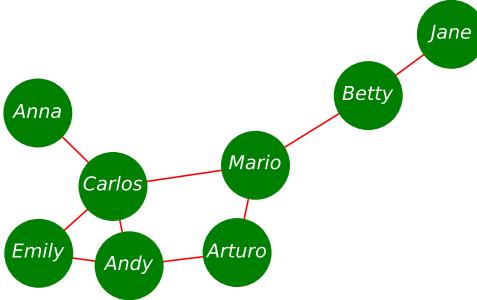


Figura 2: Relaciones de amistad en un grupo de personas.

un grafo que representa las relaciones de amistad en un grupo de diez personas. Los vértices son las personas y las aristas, las personas que tienen una relación de amistad.

Para posicionar este grafo se usó el *Fruchterman-Reingold layout*. Este fue diseñado para dibujar grafos no dirigidos, con el objetivo de distribuir los vértices uniformemente en el marco, haciendo que las longitudes de los bordes sean uniformes y reflejando la simetría [4].

```

1 import matplotlib.pyplot as plot
2 import networkx as nx
3
4 G = nx.Graph()
5 G.add_edge(1,2)
6 G.add_edge(1,3)
7 G.add_edge(1,4)
8 G.add_edge(4,5)
9 G.add_edge(2,6)
10 G.add_edge(3,8)
11 G.add_edge(7,8)
12 G.add_edge(4,8)
13 G.add_edge(4,7)
14 pos=nx.fruchterman_reingold_layout(G,scale=.45)
15
16 nx.draw_networkx_nodes(G, pos, node_size=1800, node_color='g', node_shape='o')
17 nx.draw_networkx_edges(G, pos, width=1, alpha=0.5, edge_color='red')
18
19 labels = {}
20 labels[1] = r'$ Mario $'
21 labels[2] = r'$Betty$'
22 labels[3] = r'$Arturo$'
23 labels[4] = r'$Carlos$'
24 labels[5] = r'$Anna$'
25 labels[6] = r'$Jane$'
26 labels[7] = r'$Emily$'
27 labels[8] = r'$Andy$'
28
29 nx.draw_networkx_labels(G, pos, labels, font_size=12, font_color='w')
30
31 plot.axis('off')
32 plot.savefig("fig2.eps")
33
34 plot.show()

```

Ej2.py

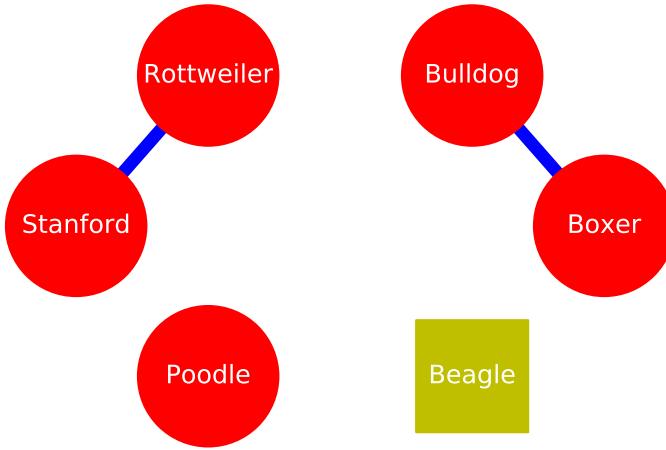


Figura 3: Estado de cría de razas de perro

### 3. Grafo simple no dirigido reflexivo

Un criador de perros posee seis razas diferentes de estos animales. Con un grafo de este tipo puede representar de cuales de estos animales ha obtenido crías, siendo los vértices las razas de perro y las aristas la unión entre razas que han tenido crías.

En la figura 3 se puede observar que todas las razas excepto la beagle, que es recién adquirida y aún no se ha reproducidos, han tenido crías con su misma raza (vértices en rojo). También se observa que han habido cruzamientos entre boxer y bulldog y entre pitbull y rottweiler.

Para posicionar este grafo se usó el *Kamada-Kawai layout*, diseñado para dibujar grafos ponderados no dirigidos [1]. Los *layouts circular* y *shell* mostraron también resultados similares.

```

1 import matplotlib.pyplot as plot
2 import networkx as nx
3
4 G = nx . Graph ()
5
6 G.add_edge(1 ,2)
7 G.add_edge(3 ,4)
8 G.add_node(5)
9 G.add_node(6)
10 apareados={1,2,3,4,5}
11 noApareado={6}
12 pos = nx.kamada_kawai_layout(G, scale=.45)
13
14 nx.draw_networkx_nodes(G, pos ,nodelist=apareados ,node_size=4400, node_color='r' ,
15   node_shape='o')
15 nx.draw_networkx_nodes(G, pos ,nodelist=noApareado , node_size=2800, node_color='y' ,
16   node_shape='s')
16 nx.draw_networkx_edges(G, pos ,width=6, alpha=0.6, edge_color='b')
17
18 labels = {}
19 labels[1] = r'Boxer'
20 labels[2] = r'Bulldog'
21 labels[3] = r'Rottweiler'
22 labels[4] = r'Stanford'
```

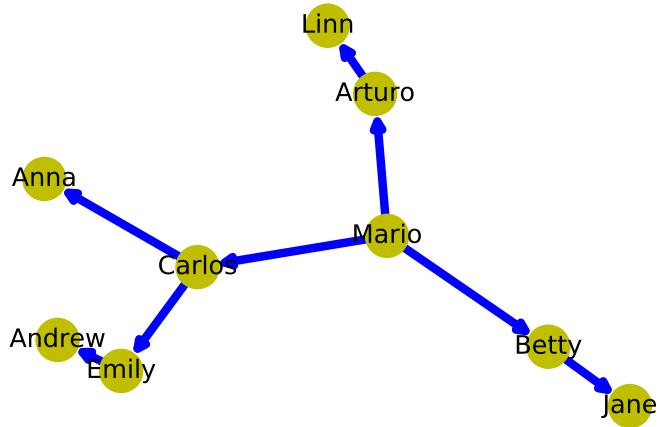


Figura 4: Representación de la transmisión de una ITS en un grupo de personas.

```

23| labels[5] = r'Poodle'
24| labels[6] = r'Beagle'
25|
26| nx.draw_networkx_labels(G, pos, labels, font_size=12, font_color='w')
27|
28| plot.axis('off')
29| plot.savefig("fig3.eps")
30| plot.show()

```

Ej3.py

#### 4. Grafo simple dirigido acíclico

La cadena de propagación de las ITS (Infecciones de Transmisión Sexual) para las cuales no se conoce cura o las que solo afectan al individuo una vez en la vida, pueden representarse mediante grafos simples dirigidos acíclicos. En este caso cada sujeto es un vértice y la dirección de transmisión de la enfermedad es representada en la arista. La enfermedad se propaga desde la persona portadora hacia el sujeto sano, que posteriormente se convierte en portador y contagia a otros sujetos sanos.

De los *layouts* empleados para posicionar los elementos de la figura 4 el Fruchterman-Reingold fue el más adecuado.

```

1 import matplotlib.pyplot as plot
2 import networkx as nx
3
4 G = nx . DiGraph ()
5 G.add_edge(1 ,2)
6 G.add_edge(1 ,3)
7 G.add_edge(1 ,4)
8 G.add_edge(4 ,5)
9 G.add_edge(2 ,6)
10 G.add_edge(7 ,8)
11 G.add_edge(4 ,7)
12 G.add_edge(3 ,9)
13 pos = nx.fruchterman_reingold_layout(G, scale=.55, iterations=30)
14 nx.draw_networkx_nodes(G, pos, node_size=400, node_color='y', node_shape='o')
15 nx.draw_networkx_edges(G, pos, width=4, edge_color='b')
16
17 labels = {}
18 labels[1] = r'Mario'
19 labels[2] = r'Betty'
20 labels[3] = r'Arturo'
21 labels[4] = r'Carlos'
22 labels[5] = r'Anna'
23 labels[6] = r'Jane'
24 labels[7] = r'Emily'
25 labels[8] = r'Andrew'
26 labels[9] = r'Linn'
27
28 nx.draw_networkx_labels(G, pos, labels, font_size=12)
29 plot.axis('off')
30 plot.savefig("fig4 .eps")
31 plot.show()

```

Ej4.py

## 5. Grafo simple dirigido cíclico

En ciudades con carreteras estrechas como Santiago de Cuba, en diferentes sectores, las carreteras son de un solo sentido y se representan con grafos de este tipo. Las aristas representan las calles y los vértices son las intersecciones entre al menos dos aristas.

Luego de probar varios *layouts*, el *spectral* es el que arroja el resultado deseado para dibujar la figura 5.

```

1 import matplotlib.pyplot as plot
2 import networkx as nx
3
4 G = nx . DiGraph ()
5
6 G.add_edge(1 ,2 , calle='c')
7 G.add_edge(2 ,3 , calle='c')
8 G.add_edge(5 ,4 , calle='b')
9 G.add_edge(6 ,5 , calle='b')
10 G.add_edge(8 ,9 , calle='a')
11 G.add_edge(7 ,8 , calle='a')
12 G.add_edge(1 ,4 , calle='19')
13 G.add_edge(4 ,7 , calle='19')
14 G.add_edge(8 ,5 , calle='21')
15 G.add_edge(5 ,2 , calle ='21')
16 G.add_edge(3 ,6 , calle ='23')
17 G.add_edge(6 ,9 , calle ='23')

```

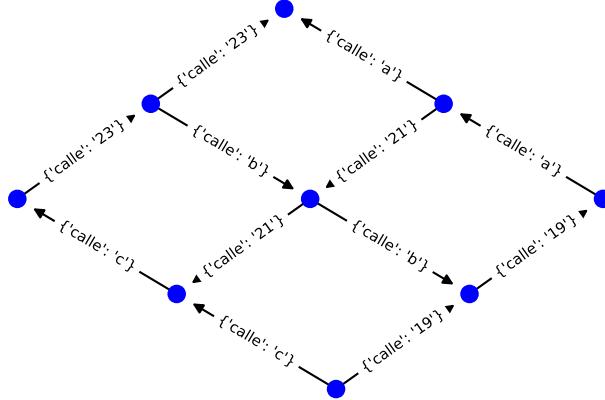


Figura 5: Carreteras de sentido único en Santiago de Cuba.

```

18 pos = nx.spectral_layout(G)
19 nx.draw_networkx_nodes(G, pos, node_size=60, node_color='b', node_shape='o')
20 nx.draw_networkx_edges(G, pos, width=1, edge_color='black')
21 edge_labels=nx.draw_networkx_edge_labels(G,pos,font_size=7,label_pos=.5)
22
23
24
25 plot.axis('off')
26 plot.savefig("fig5.eps")
27 plot.show()

```

Ej5.py

## 6. Grafo simple dirigido reflexivo

Un grafo en el que cada vértice es una empresa de servicios y las aristas, la unión con cada una de las otras empresas a las que le presta servicios, puede representarse mediante un grafo dirigido reflexivo, pues algunas de estas empresas se brindan servicio a ellas mismas.

Un sitio web pequeño, en el cual se accede a todas sus páginas a través de un menú estático también se puede representar con este tipo de grafo.

Otro ejemplo está dado por la representación de un grupo de personas conectadas a una red social y los perfiles de otras personas que tengan abiertos en el navegador en ese momento. Cada una de esas persona también pudiesen estar mirando su propio perfil. En el grafo de la figura 6 los vértices son las personas y las aristas indican el perfil de qué personas tiene cada uno abierto en el navegador. Puede apreciarse en color rojo que Mayra y Andrew tienen abiertos sus propios perfiles.

Para obtener la forma deseada del grafo ofrecen resultados similares el *circular layout* y el *shell*.

```

1 import matplotlib.pyplot as plot
2 import networkx as nx
3
4 G = nx . DiGraph ()
5

```

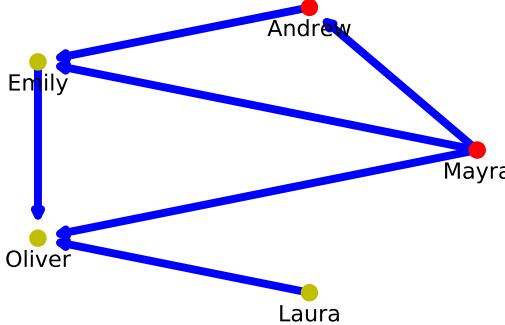


Figura 6: Representación de personas mirando perfiles en redes sociales.

```

6| G.add_edge(1,2)
7| G.add_edge(1,3)
8| G.add_edge(1,5)
9| G.add_edge(2,3)
10| G.add_edge(3,5)
11| G.add_edge(4,5)
12| node1 = {1,2}
13| node2 = {3,4,5}
14|
15|
16| pos = nx.circular_layout(G)
17|
18| nx.draw_networkx_nodes(G, pos, nodelist=node1, node_size=100, node_color='r',
19|                         node_shape='o', alpha=0.8)
20| nx.draw_networkx_nodes(G, pos, nodelist=node2, node_size=100, node_color='y',
21|                         node_shape='o')
22| nx.draw_networkx_edges(G, pos, width=5, alpha=0.4, edge_color='b')
23|
24| labels = []
25| labels[1] = r'Mayra'
26| labels[2] = r'Andrew'
27| labels[3] = r'Emily'
28| labels[4] = r'Laura'
29| labels[5] = r'Oliver'
30|
31| pos1 = pos
32| for i in pos:
33|     pos1[i][1] = pos1[i][1] - 0.15
34| nx.draw_networkx_labels(G, pos1, labels, font_size=14)
35|
36| plot.axis('off')
plot.savefig("fig6.eps")
plot.show()

```

Ej6.py

## 7. Multigrafo no dirigido acíclico

La ruta entre Santiago de Cuba y Holguín puede representarse como un multigrafo no dirigido cíclico. En un pueblo llamado Caballería la carretera principal se bifurca, pudiendo continuar por la ruta principal hasta Banes, o por el camino alternativo de Caballería, que, aunque es más largo,

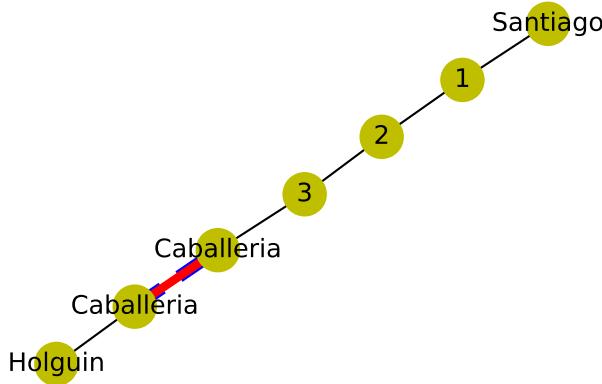


Figura 7: Posible ruta entre Santiago de Cuba y Holguín

puede recorrerse en menor tiempo a causa del poco tránsito.

En la figura 7 los vértices representan las cabeceras municipales, que están simplemente enumeradas por no ser de interés para el ejemplo mencionado. Para la visualización se empleó el *layout Kamada-Kawai*.

```

1 import matplotlib.pyplot as plot
2 import networkx as nx
3
4 G = nx . MultiGraph ()
5
6
7 G.add_edge(1,2 , weight=1)
8 G.add_edge(2,3 , weight=1)
9 G.add_edge(3,4 , weight=1)
10 G.add_edge(4,5 , weight=1)
11 G.add_edge(5,6 , weight=1)
12 G.add_edge(5,6 , weight=2)
13 G.add_edge(6,7 , weight=1)
14 black=[(1,2),(2,3),(3,4),(4,5),(6,7)]
15 blue=[(5,6)]
16 red=[(5,6)]
17 pos = nx.kamada_kawai_layout(G, scale=.2)
18
19 nx.draw_networkx_nodes(G, pos ,node_size=400, node_color='y', node_shape='o')
20 nx.draw_networkx_edges(G, pos ,edgelist=black ,width=1,edge_color='black' , alpha=0.8)
21 nx.draw_networkx_edges(G, pos , edgelist=blue ,width=6, alpha=0.5,
22 edge_color='b' , style='dashed')
23 nx.draw_networkx_edges(G, pos , edgelist=red ,width=4, alpha=0.5,
24 edge_color='r')
25 labels = {}
26 labels [1] = r'Santiago'
27 labels [2] = r'1'
28 labels [3] = r'2'
29 labels [4] = r'3'
30 labels [5] = r'Caballería'
31 labels [6] = r'Caballería'
32 labels [7] = r'Holguín'
33
34 nx.draw_networkx_labels(G, pos , labels , font_size=12)
35

```

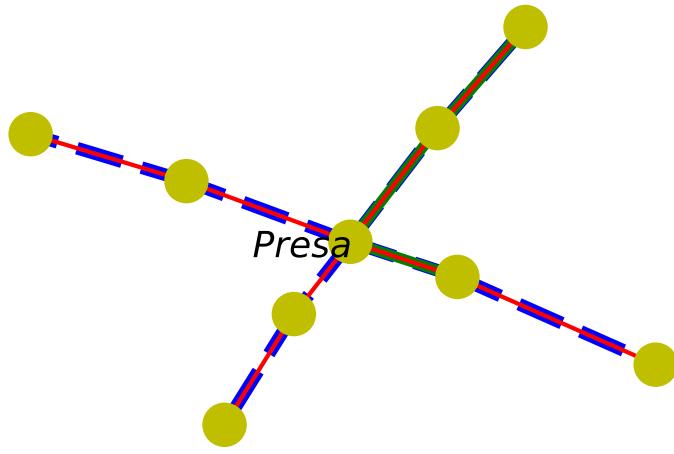


Figura 8: Canales para irrigación del suelo creados en la presa El cacao.

```

36 | plot.axis('off')
37 | plot.savefig("fig7.eps")
38 | plot.show()

```

Ej7.py

## 8. Multigrafo no dirigido cíclico

En la presa El cacao, ubicada en el Municipio Cotorro, Ciudad de La Habana, se realizó un proyecto para favorecer la agricultura. Al rededor de la presa se construyeron una serie de canales para mantener el suelo húmedo durante todo el año. Este sistema puede ser representado mediante el grafo de la figura 8, el cual los vértices representan la unión entre uno o más canales y las aristas, cada uno de los canales. El *layout Kamada-Kawai* fue el que mejores resultados de visualización arrojó.

```

1 import matplotlib.pyplot as plot
2 import networkx as nx
3
4 G = nx . MultiGraph ()
5
6 G.add_edge(1 ,2 , weight=3)
7 G.add_edge(1 ,2 , weight=5)
8 G.add_edge(1 ,2 , weight=6)
9 G.add_edge(2 ,3 , weight=4)
10 G.add_edge(2 ,3 , weight=3)
11 G.add_edge(2 ,3 , weight=5)
12 G.add_edge(1 ,4 , weight=4)
13 G.add_edge(1 ,4 , weight=3)
14 G.add_edge(4 ,5 , weight=5)
15 G.add_edge(4 ,5 , weight=3)
16 G.add_edge(1 ,6 , weight=3)
17 G.add_edge(1 ,6 , weight=2)
18 G.add_edge(6 ,7 , weight=3)
19 G.add_edge(6 ,7 , weight=4)
20 G.add_edge(1 ,8 , weight=5)
21 G.add_edge(1 ,8 , weight=4)
22 G.add_edge(1 ,8 , weight=2)
23 G.add_edge(8 ,9 , weight=5)
24 G.add_edge(8 ,9 , weight=4)
25
26 blue=[(1 ,2) ,(2 ,3) ,(1 ,4) ,(4 ,5) ,(1 ,6) ,(6 ,7) ,(1 ,8) ,(8 ,9) ]
27 red=[(1 ,2) ,(2 ,3) ,(1 ,4) ,(4 ,5) ,(1 ,6) ,(6 ,7) ,(1 ,8) ,(8 ,9) ]
28 green=[(1 ,2) ,(2 ,3) ,(1 ,8) ]
29
30 pos = nx.kamada_kawai_layout(G)
31
32 nx.draw_networkx_nodes(G, pos ,node_size=400, node_color='y' , node_shape='o')
33 nx.draw_networkx_edges(G, pos , edgelist=blue ,width=6, alpha=0.5,
34 edge_color='b' , style='dashed')
35 nx.draw_networkx_edges(G, pos , edgelist=green ,width=5, alpha=0.5,
36 edge_color='g')
37 nx.draw_networkx_edges(G, pos , edgelist=red ,width=2, alpha=0.5,
38 edge_color='r')
39 labels = {}
40 labels [1] = r'$Presa$'
41 for i in pos:
42     pos [i][0] = pos [i][0] -0.15
43     pos [i][1] = pos [i][1] -0.03
44 nx.draw_networkx_labels(G, pos , labels , font_size=17)
45
46 plot.axis('off')
47 plot.savefig("fig8 .eps")
48 plot.show()

```

Ej8.py

## 9. Multigrafo no dirigido reflexivo

En la Universidad de Oriente se quiere realizar un concurso de habilidades entre las carreras de Ingeniería en Electrónica, Ingeniería en Automática, Licenciatura en Física, Ingeniería en Informática y Licenciatura en Matemática-Cibernética. Las habilidades a evaluar serán matemática y programación.

La figura 9 muestra como está organizado el concurso. Cada vértice representa a los estudiantes que estudian una de las carreras y las aristas, con qué estudiantes pueden participar en cada habilidad.

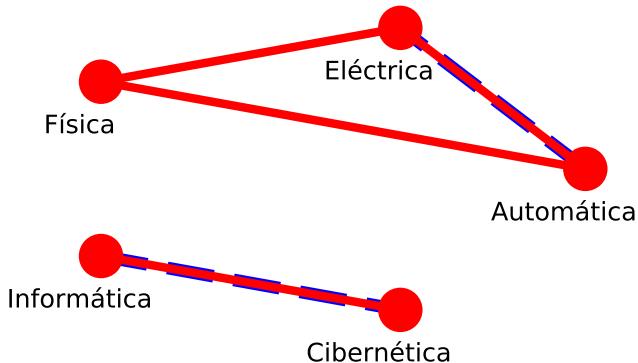


Figura 9: Concurso de habilidades.

Todos los vértices son de color rojo porque todos los estudiantes pueden competir con estudiantes de su misma carrera.

Los vértices rojos unen a los que pueden concursar entre sí en matemáticas y los azules, los que pueden concursar en programación.

Para la visualización se emplea el *layout circular*.

```

1 import matplotlib.pyplot as plot
2 import networkx as nx
3
4 G = nx . MultiGraph ()
5
6 G.add_edge(1,2, weight=1)
7 G.add_edge(1,2, weight=3)
8 G.add_edge(2,3, weight=3)
9 G.add_edge(1,3, weight=3)
10 G.add_edge(4,5, weight=1)
11 G.add_edge(4,5, weight=3)
12
13 blue=[(1,2),(4,5)]
14 red=[(1,2),(2,3),(1,3),(4,5)]
15
16 pos = nx.circular_layout(G, scale=0.1)
17
18 nx.draw_networkx_nodes(G, pos ,node_size=400, node_color='r', node_shape='o')
19 nx.draw_networkx_edges(G, pos , edgelist=blue, width=6, alpha=0.5,
20 edge_color='b', style='dashed')
21
22 nx.draw_networkx_edges(G, pos , edgelist=red, width=4, alpha=0.5,
23 edge_color='r')
24 labels = {}
25 labels [1] = r'Automatica'
26 labels [2] = r'Electrica'
27 labels [3] = r'Fisica'
28 labels [4] = r'Informatica'
29 labels [5] = r'Cibernetica'
30
31 for i in pos:
32     pos [i][1] = pos [i][1] -0.03
33     pos [i][0] = pos [i][0] -0.008

```

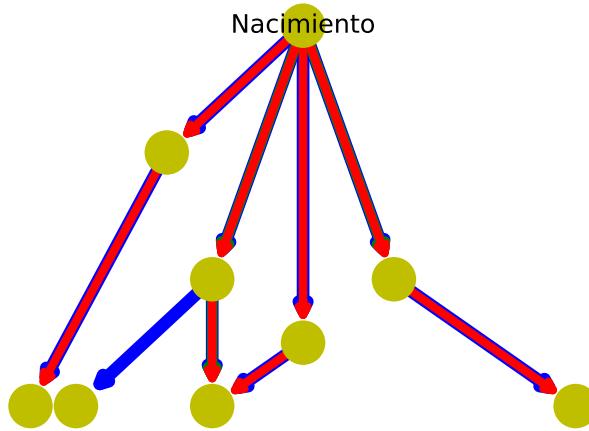


Figura 10: Representación del delta de un río en su transcurso hacia el mar.

```

34 nx.draw_networkx_labels(G, pos, labels, font_size=12)
35
36 plot.axis('off')
37 plot.savefig("fig9.eps")
38 plot.show()

```

Ej9.py

## 10. Multigrafo dirigido acíclico

Estos grafos pueden utilizarse para representar la cuenca hidrográfica de un río en su flujo hacia el mar. En la figura 10 los vértices representan los puntos en los que al menos dos ramificaciones del delta confluyen o se separan. Para que el grafo tuviese exactamente la forma deseada, se insertaron las coordenadas de los vértices de manera manual.

```

1 import matplotlib.pyplot as plot
2 import networkx as nx
3
4 G = nx.MultiDiGraph()
5
6 G.add_edge(1,2, weight=3)
7 G.add_edge(1,2, weight=5)
8 G.add_edge(1,2, weight=6)
9 G.add_edge(2,3, weight=4)
10 G.add_edge(2,5, weight=3)
11 G.add_edge(2,5, weight=5)
12 G.add_edge(1,4, weight=4)
13 G.add_edge(1,4, weight=3)
14 G.add_edge(4,5, weight=5)
15 G.add_edge(4,5, weight=3)
16 G.add_edge(1,6, weight=3)
17 G.add_edge(1,6, weight=2)
18 G.add_edge(6,7, weight=1)
19 G.add_edge(6,7, weight=4)
20 G.add_edge(1,8, weight=5)
21 G.add_edge(1,8, weight=4)
22 G.add_edge(1,8, weight=2)

```

```

23 G.add_edge(8,9, weight=5)
24 G.add_edge(8,9, weight=4)
25
26 blue=[(1,2),(2,3),(2,5),(1,4),(4,5),(1,6),(6,7),(1,8),(8,9)]
27 red=[(1,2),(2,5),(1,4),(4,5),(1,6),(6,7),(1,8),(8,9)]
28 green=[(1,2),(2,5),(1,8)]
29
30 pos = {1:(400, 700), 2:(300,300), 3:(150, 100), 4:(400,200), 5:(300,100),6:(250,500),
31 , 7:(100
32 ,100), 8:(500, 300), 9:(700,100)}
33 nx.draw_networkx_nodes(G, pos, node_size=400, node_color='y', node_shape='o')
34 nx.draw_networkx_edges(G, pos, edgelist=blue, width=6, alpha=0.5,
35 edge_color='b', style='dashed')
36 nx.draw_networkx_edges(G, pos, edgelist=green, width=5, alpha=0.5,
37 edge_color='g')
38 nx.draw_networkx_edges(G, pos, edgelist=red, width=4, alpha=0.5,
39 edge_color='r')
40 labels = {}
41 labels[1] = r'Nacimiento'
42
43 nx.draw_networkx_labels(G, pos, labels, font_size=12)
44
45 plot.xlim(20,800)
46 plot.axis('off')
47 plot.savefig("fig10.eps")
48 plot.show()

```

Ej10.py

## 11. Multigrafo dirigido cíclico

Este tipo de grafo es especialmente útil para representar viajes por lugares a los que se puede llegar mediante diferentes vías, regresando luego al punto de origen. Un ejemplo concreto de esto sería un recorrido vacacional por varias ciudades. Partiendo de la ciudad A, hay tres posibles vías para llegar a la ciudad B, en bus, en auto de alquiler o en tren, cada uno de estos medios de transporte representaría una arista diferente para unir los vértices. Así sucesivamente, se tienen en cuenta las posibles vías de transporte hacia las distintas ciudades hasta completar el recorrido y regresar a casa.

Para posicionar los vértices en la figura 11 se emplea el *layout shell*, aunque el circular también ofrece buenos resultados en este tipo de grafos.

```

1 import matplotlib.pyplot as plot
2 import networkx as nx
3
4 G = nx . MultiDiGraph ()
5 G.add_edge(1,2, weight=3)
6 G.add_edge(1,2, weight=5)
7 G.add_edge(1,2, weight=6)
8 G.add_edge(2,3, weight=3)
9 G.add_edge(2,3, weight=5)
10 G.add_edge(2,3, weight=6)
11 G.add_edge(3,4, weight=3)
12 G.add_edge(4,5, weight=3)
13 G.add_edge(4,5, weight=5)
14 G.add_edge(5,6, weight=3)
15 G.add_edge(6,7, weight=6)
16 G.add_edge(7,1, weight=6)
17
18 blue=[(1,2),(2,3),(3,4),(4,5),(5,6)]

```

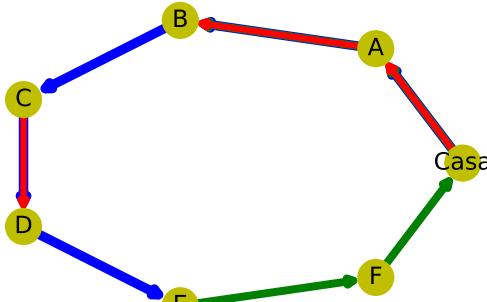


Figura 11: Representación de las posibles maneras de elegir la ruta de un circuito vacacional.

```

19 red=[(1,2),(2,3),(4,5)]
20 green=[(1,2),(2,3),(6,7),(7,1)]
21
22 pos = nx.shell_layout(G)
23
24 nx.draw_networkx_nodes(G, pos ,node_size=500, node_color='y', node_shape='o')
25 nx.draw_networkx_edges(G, pos , edgelist=blue ,width=6, alpha=0.5,
26 edge_color='b', style='dashed')
27 nx.draw_networkx_edges(G, pos , edgelist=green ,width=5, alpha=.7,
28 edge_color='g')
29 nx.draw_networkx_edges(G, pos , edgelist=red ,width=4, alpha=0.5,
30 edge_color='r')
31
32 labels = {}
33 labels[1] = r'Casa',
34 labels[2] = r'A',
35 labels[3] = r'B',
36 labels[4] = r'C',
37 labels[5] = r'D',
38 labels[6] = r'E',
39 labels[7] = r'F',
40
41 nx.draw_networkx_labels(G, pos , labels , font_size=15, )
42 plot.axis('off')
43 plot.savefig("fig11.eps")
44 plot.show()

```

Ej11.py

## 12. Multigrafo dirigido reflexivo

González-Cervantes [5] emplea teoría de grafos para representar el potencial eléctrico en el corazón, demostrando que se pueden incorporar las leyes fisiológicas involucradas. Cada uno de los vértices representa uno de los puntos principales que generan los impulsos eléctricos y que llevan la electricidad a cada parte del corazón, las aristas describen el valor máximo de voltaje y su duración en tiempo que descarga cada vértice. Además, puede proporcionar información con respecto al potencial eléctrico por zonas para una mejor localización. En la figura 12 se muestra la representación con grafos del intervalo PR. Debido a que se siguió la forma del corazón para dibujar el grafo, las coordenadas de los vértices fueron ingresadas manualmente.

```

1 import matplotlib.pyplot as plot

```

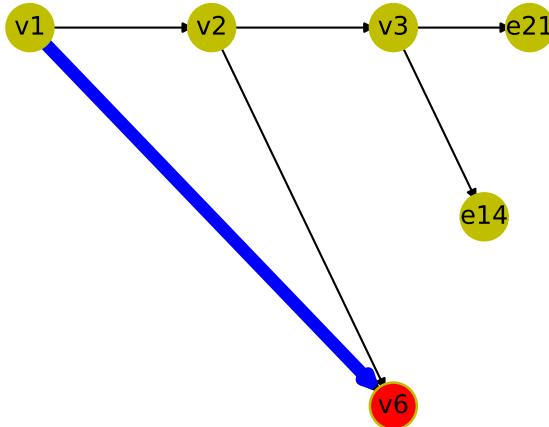


Figura 12: Representación con grafos del intervalo PR.

```

2 import networkx as nx
3
4 G = nx . MultiDiGraph ()
5
6 G.add_edge(1 ,2)
7 G.add_edge(2 ,3)
8 G.add_edge(3 ,4)
9 G.add_edge(3 ,5)
10 G.add_edge(1 ,6 , weight=3)
11 G.add_edge(1 ,6 , weight=1)
12 blue=[(1 ,6)]
13 G.add_edge(2 ,6)
14 node1 = {6}
15
16 pos = {1:(50 , 350) , 2:(250 ,350) , 3:(450 , 350) , 4:(600 ,350) , 5:(550 ,340) , 6:(450 ,330)
17 }
18 nx.draw_networkx_nodes(G, pos , node_size=500, node_color='y' , node_shape='o')
19 nx.draw_networkx_edges(G, pos , width=1, alpha=0.8, edge_color='black')
20 nx.draw_networkx_nodes(G, pos , nodelist=node1 , node_size=400, node_color='r' ,
21 node_shape='o')
22 nx.draw_networkx_edges(G, pos , edgelist=blue ,width=6, alpha=0.5,
23 edge_color='b' , style='dashed')
24
25 labels = {}
26 labels [1] = r 'v1'
27 labels [2] = r 'v2'
28 labels [3] = r 'v3'
29 labels [4] = r 'e21'
30 labels [5] = r 'e14'
31 labels [6] = r 'v6'
32
33 nx.draw_networkx_labels(G, pos , labels , font_size=12)
34 plot.xlim(20 ,800)
35 plot.axis('off')
36 plot.savefig("fig12 .eps")
37 plot.show()

```

Ej12.py

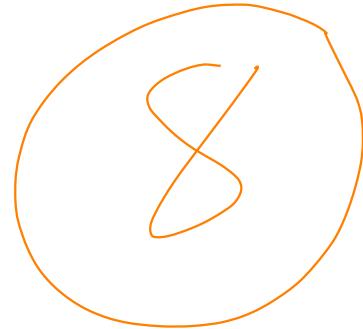
## Referencias

- [1] An algorithm for drawing general y otros. *Information processing letters*, 31(1):7–15, 1989.
- [2] Ravindra K Ahuja. *Network flows: theory, algorithms, and applications*. Pearson Education, 2017.
- [3] Fan Chung. Graph theory in the information age. *Notices of the AMS*, 57(6):726–732, 2010.
- [4] Edward M Fruchterman, Thomas MJ y Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.
- [5] Aurora y Salido-Ruiz Ricardo González-Cervantes, Natalia y Espinoza-Valdez. Potencial eléctrico en el corazón: Representación mediante un grafo. *ReCIBE*, 5(3), 2016.
- [6] Amador Menéndez Velázquez. Una breve introducción a la teoría de grafos. *Suma*, 28:11–26, 1998.

## Tarea 2

5280

19 de marzo de 2019



## Introducción

En este trabajo se realizan mediciones sobre los algoritmos para grafos de NetworkX [2]:

- *betweenness\_centrality*: Ofrece una medida de centralidad de un grafo, la cual devuelve como un diccionario de nodos, con la medida de centralidad. Puede usarse en grafos dirigidos y no dirigidos.
- *maximal\_matching*: Devuelve un conjunto de nodos máximo conjunto posible de aristas independientes, que no tienen nodos en común.
- *greedy\_color*: Colorea los nodo usando diferentes estrategias. Devuelve un diccionario con claves que representan nodos y valores que representan la coloración. Puede emplearse en grafos dirigidos y no dirigidos.
- *make\_max\_clique\_graph*: Devuelve el subgrafo más grande que encuentra. Debe usarse en grafos no dirigidos.
- *strongly\_connected\_components*: Se utiliza para grafos dirigidos. Devuelve un generador de conjunto de nodos, uno por cada componente fuertemente conectado.

A los cuatro primeros algoritmos se les puede pasar como parámetro grafos no dirigidos ponderados, lo que resulta conveniente pues permite emplear el mismo conjuntos de grafos para los cuatro algoritmos. El algoritmo *strongly\_connected\_components* requiere grafos dirigidos, por lo que para este se emplea otro conjunto de grafos.

## Generación de grafos

Todos los grafos son generados aleatoriamente empleando las funciones **GenerateGraph** y **GenerateDiGraph**. A estas funciones se le pasan los siguientes parámetros:

*Verdadero*

- *nameToSave*: Prefijo de la dirección con la que se quieren guardar los grafos. El código añadirá números consecutivos a este prefijo para cada grafo, comenzando por 0.
- *Smin*: Número mínimo de nodos que tendrán los gráficos a generar.
- *Smax*: Número máximo de nodos que tendrán los gráficos a generar.

- *max\_weight*: Peso máximo que tendrán las aristas.
- *numberOfGraphs*: Cantidad de grafos a generar.

Los grafos son convertidos a *DataFrame* de la librería pandas [3] y guardados en formato *.csv*.

```

1 def GenerateGraph(nameToSave, Smin, Smax, max_weight, numberOfGraphs):
2     for h in range(numberOfGraphs):
3         G=nx.Graph()
4         size=rdm.randint(Smin,Smax)
5         for i in range(size):
6             for j in range(i, size):
7                 if rdm.randint(0,int(size/20))==0:
8                     G.add_edge(i,j, weight=rdm.randint(1,max_weight))
9         df = pd.DataFrame()
10        df = nx.to_pandas_adjacency(G, dtype=int, weight='weight')
11        df.to_csv(nameToSave+str(h)+".csv")
12
13 def GenerateDiGraph(nameToSave, Smin, Smax, max_weight, numberOfGraphs):
14     for h in range(numberOfGraphs):
15         G=nx.DiGraph()
16         size=rdm.randint(Smin,Smax)
17         for i in range(size):
18             for j in range(i, size):
19                 if rdm.randint(0,int(size/50))==0:
20                     G.add_edge(i,j, weight=rdm.randint(1,max_weight))
21                 elif rdm.randint(0,int(size/30))==1:
22                     G.add_edge(j,i, weight=rdm.randint(1,max_weight))
23         df = pd.DataFrame()
24         df = nx.to_pandas_adjacency(G, dtype=int, weight='weight')
25         df.to_csv(nameToSave+str(h)+".csv")

```

Tarea3.py

## Medición del tiempo de ejecución

Para cada uno de los algoritmos seleccionados se realiza una función que recibe un grafo como parámetro, ejecuta el algoritmo y devuelve el tiempo de ejecución del mismo.

```

1 def BetCen(graph):
2     start_time=time()
3     R = nx.betweenness_centrality(graph, weight='size', normalized=False)
4     time_elapsed = time() - start_time
5     return time_elapsed
6
7 def MinMaxMat(graph):
8     start_time=time()
9     for i in range(100):
10        R = nx.maximal_matching(graph)
11        time_elapsed = time() - start_time
12        return time_elapsed
13
14 def GreedyColor(graph):
15     start_time=time()
16     for i in range(160):
17         R = nx.greedy_color(graph, strategy='largest_first', interchange=False)
18         time_elapsed = time() - start_time
19         return time_elapsed
20
21 def MaxClique(graph):
22     start_time=time()

```

```

13     R = nx.make_max_clique_graph(graph, create_using=None)
14     time_elapsed = time() - start_time
15     return time_elapsed
16
17 def StronglyC(graph):
18     start_time=time()
19     for i in range(99000):
20         R = nx.strongly_connected_components(graph)
21     time_elapsed = time() - start_time
22     return time_elapsed

```

Tarea3.py

Estas funciones son ejecutadas mediante **RunAll**, función a la que se le pasan los siguientes parámetros:

- *runs*: Número de veces que se quieren ejecutar los algoritmos.
- *numAlgorithms*: Cantidad de algoritmos a emplear.
- *numGraphs*: Cantidad de grafos que se va a correr para cada algoritmo.
- *name*: Formato del nombre con el que fueron guardados los grafos no dirigidos.
- *nameDi*: Formato de nombre con el que fueron guardados los grafos dirigidos.
- *matrix*: Este parámetro es un diccionario vacío cuyos índices son los números del 0 al 24.

```

1 def RunAll(runs , numAlgorithms , numGraphs , name , nameDi , matrix ) :
2     combinations =[]
3     for i in range(numAlgorithms-1):
4         for j in range (numGraphs):
5             combinations.append ([ i ,name+str (j)+".csv" ])
6     for j in range (numGraphs):
7         combinations.append ([ numAlgorithms-1 ,nameDi+str (j)+".csv" ])
8     for j in range(runs):
9         np.random.shuffle(combinations)
10        for i in combinations:
11            if i [0]==0:
12                for n in range(numGraphs):
13                    if i [1]==name+str (n)+".csv" :
14                        matrix [ str (n) ].append(BetCen(ReadGraph(i [1] )))
15            if i [0]==1:
16                for n in range(numGraphs):
17                    if i [1]== name+str (n)+".csv" :
18                        matrix [ str (n+5) ].append(MinMaxMat(ReadGraph(i [1] )))
19            if i [0]==2:
20                for n in range(numGraphs):
21                    if i [1]== name+str (n)+".csv" :
22                        matrix [ str (n+10) ].append(GreedyColor(ReadGraph(i [1] )))
23            if i [0]==3:
24                for n in range(numGraphs):
25                    if i [1]== name+str (n)+".csv" :
26                        matrix [ str (n+15) ].append(MaxClique(ReadGraph(i [1] )))
27            if i [0]==4:
28                for n in range(numGraphs):
29                    if i [1]== "directed"+str (n)+".csv" :
30                        matrix [ str (n+20) ].append(StronglyC(ReadDiGraph(i [1] )))
31
32 df = pd.DataFrame(matrix)
33 df.to_csv("matrix.csv")

```

Tarea3.py

Este código crea una lista con las 25 combinaciones de algoritmo-grafo y para cada repetición cambia la ubicación de los elementos de la lista para garantizar la aleatoricidad del proceso. Una vez terminadas todas las mediciones, son guardadas en el archivo *Matrix.csv*

~~Matrix.csv~~

## Cálculo de parámetros

Con la función **MediaDesv** se calcula la media y la desviación estándar para cada algoritmo. Esta función extrae además el número de nodos y aristas de los grafos creados, información necesaria para la creación de las gráficas de dispersión.

```
1 def MediaDesv ( adress , runs , numberOfGraphs , numAlgorithms , name , nameDi ) :
2     media = {
3         'Media0' : [ ] ,
4         'Media1' : [ ] ,
5         'Media2' : [ ] ,
6         'Media3' : [ ] ,
7         'Media4' : [ ] ,
8     }
9     standar={ 'Standar0' : [ ] ,
10        'Standar1' : [ ] ,
11        'Standar2' : [ ] ,
12        'Standar3' : [ ] ,
13        'Standar4' : [ ] ,
14    }
15    grafos={ 'EdgesUndirected' :[ ] ,
16        'NodesUndirected' :[ ] ,
17        'EdgesDirected' :[ ] ,
18        'NodesDirected' :[ ]
19    }
20
21    matrix = pd.read_csv ( adress )
22    for n in range ( 5 ) :
23        grafos [ 'EdgesUndirected' ]. append ( ReadGraph ( name+str ( n ) +".csv" ) .
24            number_of_edges () )
25        grafos [ 'NodesUndirected' ]. append ( ReadGraph ( name+str ( n ) +".csv" ) .
26            number_of_nodes () )
27        grafos [ 'EdgesDirected' ]. append ( ReadGraph ( nameDi+str ( n ) +".csv" ) .
28            number_of_edges () )
29        grafos [ 'NodesDirected' ]. append ( ReadGraph ( nameDi+str ( n ) +".csv" ) .
30            number_of_nodes () )
31        for i in range ( 5 ) :
32            media [ 'Media'+str ( i ) ]. append ( np.mean ( matrix [ str ( n+(5*i) ) ] ) )
33            standar [ 'Standar'+str ( i ) ]. append ( np.std ( matrix [ str ( n+(5*i) ) ] ) )
34
35    df=pd.DataFrame ( media )
36    df.to_csv ( "Media.csv" , index=None )
37    df=pd.DataFrame ( standar , )
38    df.to_csv ( "Standar.csv" , index=None )
39    df=pd.DataFrame ( grafos )
40    df.to_csv ( "Grafos.csv" , index=None )
```

Tarea3.py

Estos parámetros son guardados en formato *.csv* para su posterior utilización.

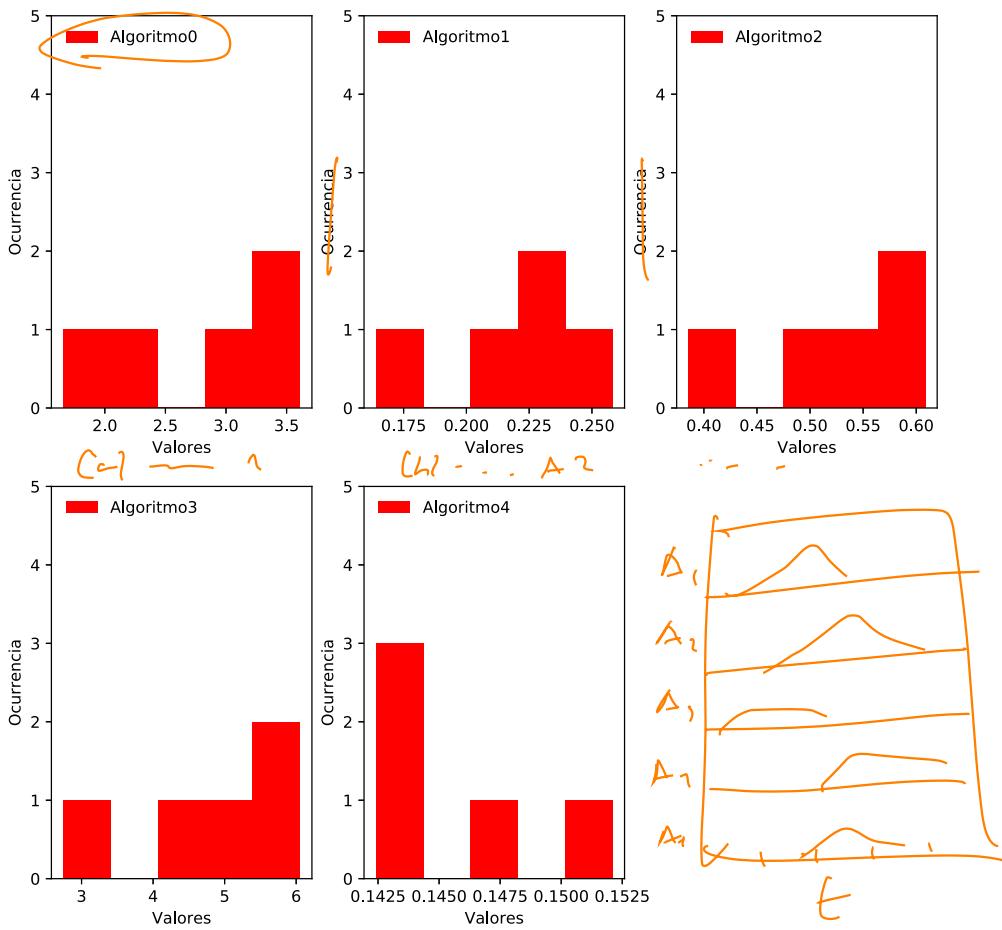


Figura 1: Histograma de las medias para cada combinación algoritmo-grafo

## Histograma

Una vez obtenidos los valores promedios de tiempo de ejecución para cada combinación algoritmo-grafo, se grafica un histograma con los resultados, empleando la librería Matplotlib[1]. En la Figura 1 se observan los histogramas correspondientes a cada algoritmo.

## Gráficas de dispersión

En la Figura 2 se observa la gráfica de dispersión en la que el eje horizontal corresponde al tiempo promedio de ejecución y el eje vertical al el número de nodos del grafo. En la gráfica de dispersión de la Figura 3, el eje horizontal corresponde de igual manera al tiempo de ejecución de los algoritmos, mientras que el vertical corresponde al número de aristas de los grafos.

Leyenda de formas:

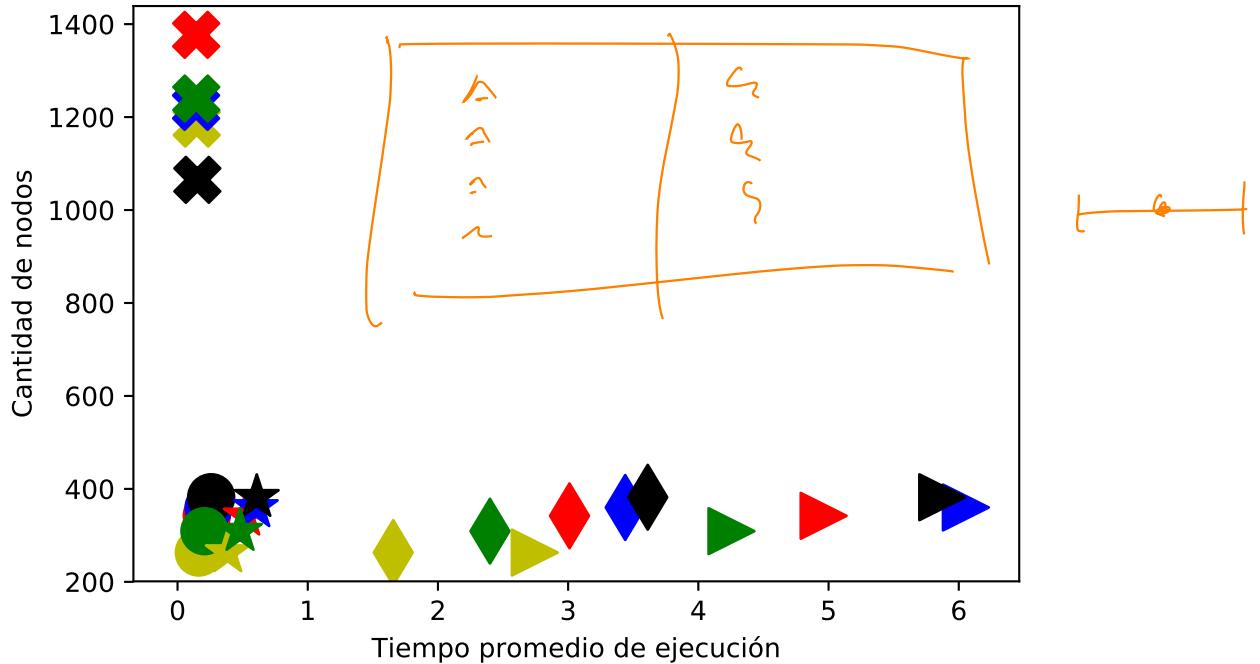
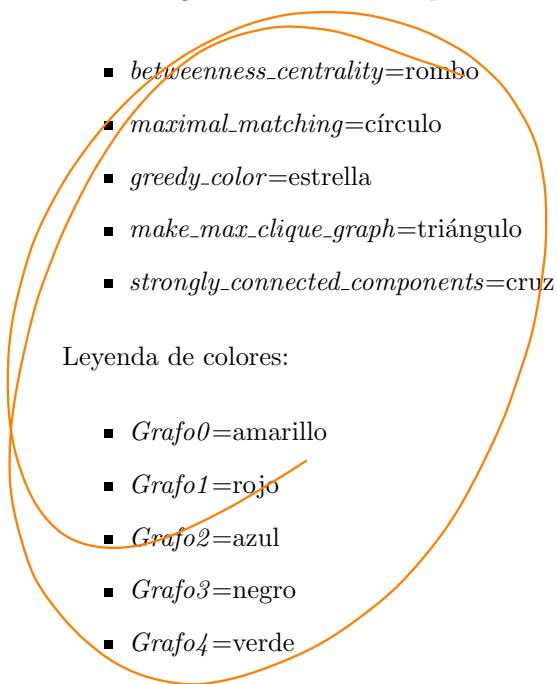


Figura 2: Gráfica de dispersión. Número de nodos respecto al tiempo de ejecución.



## Conclusiones

De manera general, pudo comprobarse que a medida que aumenta el número de nodos y/o aristas, aumenta el tiempo de ejecución de los algoritmos.

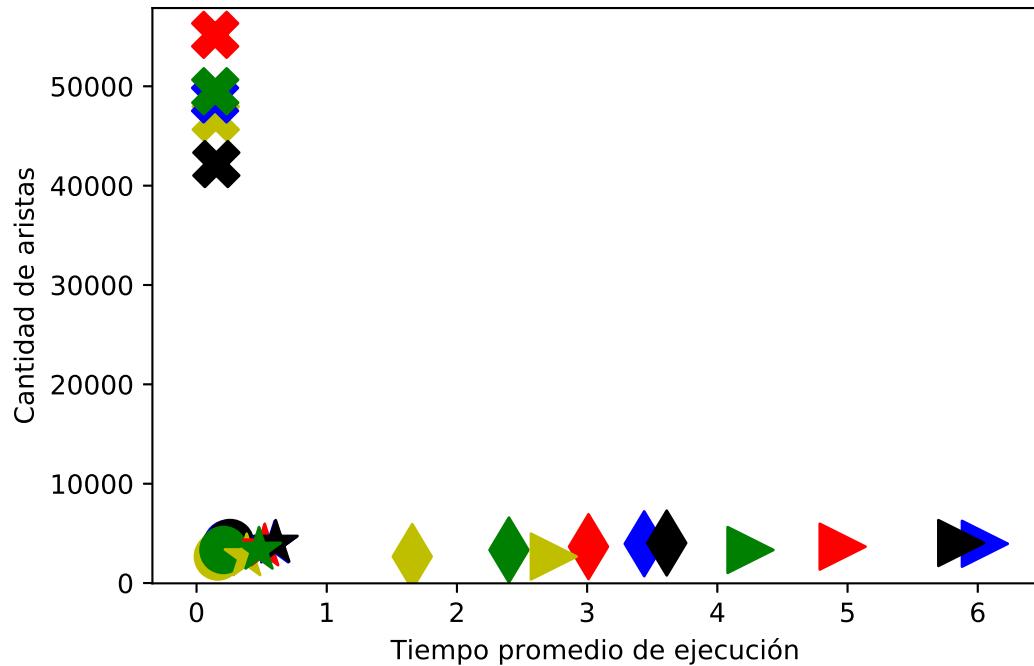


Figura 3: Gráfica de dispersión. Número de aristas respecto al tiempo de ejecución.

## Referencias

- [1] Matplotlib. url<https://matplotlib.org>. Accedido 18-3-2019.
- [2] Networkx. <https://networkx.github.io/documentation/stable/reference/algorithms/>.
- [3] Python data analysis library. url<https://pandas.pydata.org>, 2018). Accedido 18-3-2019.

# Tarea 3

5280

3 de junio de 2019

## Introducción

En este trabajo se realizan mediciones sobre los algoritmos para grafos de NetworkX [2]:

- **betweenness\_centrality**: Ofrece una medida de centralidad de un grafo, la cual devuelve como un diccionario de nodos, con la medida de centralidad. Puede usarse en grafos dirigidos y no dirigidos.
- **maximal\_matching**: Devuelve un conjunto de nodos máximo conjunto posible de aristas independientes, que no tienen nodos en común.
- **greedy\_color**: Colorea los nodo usando diferentes estrategias. Devuelve un diccionario con claves que representan nodos y valores que representan la coloración. Puede emplearse en grafos dirigidos y no dirigidos.
- **make\_max\_clique\_graph**: Devuelve el subgrafo más grande que encuentra. Debe usarse en grafos no dirigidos.
- **strongly\_connected\_components**: Se utiliza para grafos dirigidos. Devuelve un generador de conjunto de nodos, uno por cada componente fuertemente conectado.

A los cuatro primeros algoritmos se les puede pasar como parámetro grafos no dirigidos ponderados, lo que resulta conveniente pues permite emplear el mismo conjuntos de grafos para los cuatro algoritmos. El algoritmo **strongly\_connected\_components** requiere grafos dirigidos, por lo que para este se emplea otro conjunto de grafos.

## Generación de grafos

Todos los grafos son generados aleatoriamente empleando las funciones `GenerateGraph` y `GenerateDiGraph`. A estas funciones se le pasan los siguientes parámetros:

- **nameToSave**: Prefijo de la dirección con la que se quieren guardar los grafos. El código añadirá números consecutivos a este prefijo para cada grafo, comenzando por 0.
- **Smin**: Número mínimo de nodos que tendrán los gráficos a generar.
- **Smax**: Número máximo de nodos que tendrán los gráficos a generar.

- `max_weight`: Peso máximo que tendrán las aristas.
- `numberOfGraphs`: Cantidad de grafos a generar.

Los grafos son convertidos a `DataFrame` de la librería pandas [3] y guardados en formato `.csv`.

```

1 def GenerateGraph(nameToSave, Smin, Smax, max_weight, numberOfGraphs):
2     for h in range(numberOfGraphs):
3         G=nx.Graph()
4         size=rdm.randint(Smin,Smax)
5         for i in range(size):
6             for j in range(i, size):
7                 if rdm.randint(0,int(size/20))==0:
8                     G.add_edge(i,j, weight=rdm.randint(1,max_weight))
9         df = pd.DataFrame()
10        df = nx.to_pandas_adjacency(G, dtype=int, weight='weight')
11        df.to_csv(nameToSave+str(h)+".csv")
12
13 def GenerateDiGraph(nameToSave, Smin, Smax, max_weight, numberOfGraphs):
14     for h in range(numberOfGraphs):
15         G=nx.DiGraph()
16         size=rdm.randint(Smin,Smax)
17         for i in range(size):
18             for j in range(i, size):
19                 if rdm.randint(0,int(size/50))==0:
20                     G.add_edge(i,j, weight=rdm.randint(1,max_weight))
21                 elif rdm.randint(0,int(size/30))==1:
22                     G.add_edge(j,i, weight=rdm.randint(1,max_weight))
23         df = pd.DataFrame()
24         df = nx.to_pandas_adjacency(G, dtype=int, weight='weight')
25         df.to_csv(nameToSave+str(h)+".csv")

```

Tarea3.py

## Medición del tiempo de ejecución

Para cada uno de los algoritmos seleccionados se realiza una función que recibe un grafo como parámetro, ejecuta el algoritmo y devuelve el tiempo de ejecución del mismo.

```

1 def BetCen(graph):
2     start_time=time()
3     R = nx.betweenness_centrality(graph, weight='size', normalized=False)
4     time_elapsed = time() - start_time
5     return time_elapsed
6
7 def MinMaxMat(graph):
8     start_time=time()
9     for i in range(100):
10        R = nx.maximal_matching(graph)
11        time_elapsed = time() - start_time
12        return time_elapsed
13
14 def GreedyColor(graph):
15     start_time=time()
16     for i in range(160):
17         R = nx.greedy_color(graph, strategy='largest_first', interchange=False)
18         time_elapsed = time() - start_time
19         return time_elapsed
20
21 def MaxClique(graph):
22     start_time=time()

```

```

13     R = nx.make_max_clique_graph(graph, create_using=None)
14     time_elapsed = time() - start_time
15     return time_elapsed
16
17 def StronglyC(graph):
18     start_time=time()
19     for i in range(99000):
20         R = nx.strongly_connected_components(graph)
21     time_elapsed = time() - start_time
22     return time_elapsed

```

Tarea3.py

Estas funciones son ejecutadas mediante `RunAll`, función a la que se le pasan los siguientes parámetros:

- `runs`: Número de veces que se quieren ejecutar los algoritmos.
- `numAlgorithms`: Cantidad de algoritmos a emplear.
- `numGraphs`: Cantidad de grafos que se va a correr para cada algoritmo.
- `name`: Formato del nombre con el que fueron guardados los grafos no dirigidos.
- `nameDi`: Formato de nombre con el que fueron guardados los grafos dirigidos.
- `matrix`: Este parámetro es un diccionario vacío cuyos índices son los números del 0 al 24.

```

1 def RunAll(runs , numAlgorithms , numGraphs , name , nameDi , matrix ) :
2     combinations = []
3     for i in range(numAlgorithms-1):
4         for j in range (numGraphs):
5             combinations.append ([ i , name+str(j)+".csv" ])
6     for j in range (numGraphs):
7         combinations.append ([ numAlgorithms-1 , nameDi+str(j)+".csv" ])
8     for j in range(runs):
9         np.random.shuffle(combinations)
10        for i in combinations:
11            if i[0]==0:
12                for n in range(numGraphs):
13                    if i[1]==name+str(n)+".csv":
14                        matrix [ str(n) ].append(BetCen(ReadGraph(i[1])))
15            if i[0]==1:
16                for n in range(numGraphs):
17                    if i[1]== name+str(n)+".csv":
18                        matrix [ str(n) ].append(MinMaxMat(ReadGraph(i[1])))
19            if i[0]==2:
20                for n in range(numGraphs):
21                    if i[1]== name+str(n)+".csv":
22                        matrix [ str(n+10) ].append(GreedyColor(ReadGraph(i[1])))
23            if i[0]==3:
24                for n in range(numGraphs):
25                    if i[1]== name+str(n)+".csv":
26                        matrix [ str(n+15) ].append(MaxClique(ReadGraph(i[1])))
27            if i[0]==4:
28                for n in range(numGraphs):
29                    if i[1]== "directed"+str(n)+".csv":
30                        matrix [ str(n+20) ].append(StronglyC(ReadDiGraph(i[1])))
31
32 df = pd.DataFrame(matrix)
33 df.to_csv("matrix.csv")

```

Tarea3.py

Este código crea una lista con las 25 combinaciones de algoritmo-grafo y para cada repetición cambia la ubicación de los elementos de la lista para garantizar la aleatoricidad del proceso. Una vez terminadas todas las mediciones, son guardadas en el archivo **Matrix.csv**

## Cálculo de parámetros

Con la función **MediaDesv** se calcula la media y la desviación estándar para cada algoritmo. Esta función extrae además el número de nodos y aristas de los grafos creados, información necesaria para la creación de las gráficas de dispersión.

```

1 def MediaDesv ( adress , runs , numberOfGraphs , numAlgorithms , name , nameDi ) :
2     media = {
3         'Media0' : [ ] ,
4         'Media1' : [ ] ,
5         'Media2' : [ ] ,
6         'Media3' : [ ] ,
7         'Media4' : [ ] ,
8     }
9     standar={ 'Standar0' : [ ] ,
10        'Standar1' : [ ] ,
11        'Standar2' : [ ] ,
12        'Standar3' : [ ] ,
13        'Standar4' : [ ] ,
14    }
15    grafos={ 'EdgesUndirected' : [ ] ,
16        'NodesUndirected' : [ ] ,
17        'EdgesDirected' : [ ] ,
18        'NodesDirected' : [ ] ,
19    }
20
21    matrix = pd.read_csv ( adress )
22    for n in range ( 5 ) :
23        grafos [ 'EdgesUndirected' ]. append ( ReadGraph ( name + str ( n ) + ".csv" ) .
24            number_of_edges () )
25        grafos [ 'NodesUndirected' ]. append ( ReadGraph ( name + str ( n ) + ".csv" ) .
26            number_of_nodes () )
27        grafos [ 'EdgesDirected' ]. append ( ReadGraph ( nameDi + str ( n ) + ".csv" ) .
28            number_of_edges () )
29        grafos [ 'NodesDirected' ]. append ( ReadGraph ( nameDi + str ( n ) + ".csv" ) .
30            number_of_nodes () )
31        for i in range ( 5 ) :
32            media [ 'Media' + str ( i ) ]. append ( np.mean ( matrix [ str ( n + ( 5 * i ) ) ] ) )
33            standar [ 'Standar' + str ( i ) ]. append ( np.std ( matrix [ str ( n + ( 5 * i ) ) ] ) )
34
35    df=pd.DataFrame ( media )
36    df.to_csv ( "Media.csv" , index=None )
37    df=pd.DataFrame ( standar , )
38    df.to_csv ( "Standar.csv" , index=None )
39    df=pd.DataFrame ( grafos )
40    df.to_csv ( "Grafos.csv" , index=None )

```

Tarea3.py

Estos parámetros son guardados en formato **.csv** para su posterior utilización.

## Histograma

Una vez obtenidos los valores promedios de tiempo de ejecución para cada combinación algoritmo-grafo, se grafica un histograma con los resultados, empleando la librería Matplotlib[1]. En la figura 1 se observan los histogramas correspondientes a cada algoritmo.

## Gráficas de dispersión

En la figura 2 se observa la gráfica de dispersión en la que el eje horizontal corresponde al tiempo promedio de ejecución y el eje vertical al el número de nodos del grafo. En la gráfica de dispersión de la figura 3, el eje horizontal corresponde de igual manera al tiempo de ejecución de los algoritmos, mientras que el vertical corresponde al número de aristas de los grafos.

Leyenda de formas:

- `betweenness_centrality`=rombo
- `maximal_matching`=círculo
- `greedy_color`=estrella
- `make_max_clique_graph`=triángulo
- `strongly_connected_components`=cruz

Leyenda de colores:

- Grafo0=amarillo
- Grafo1=rojo
- Grafo2=azul
- Grafo3=negro
- Grafo4=verde

## Conclusiones

De manera general, pudo comprobarse que a medida que aumenta el número de nodos y/o aristas, aumenta el tiempo de ejecución de los algoritmos.

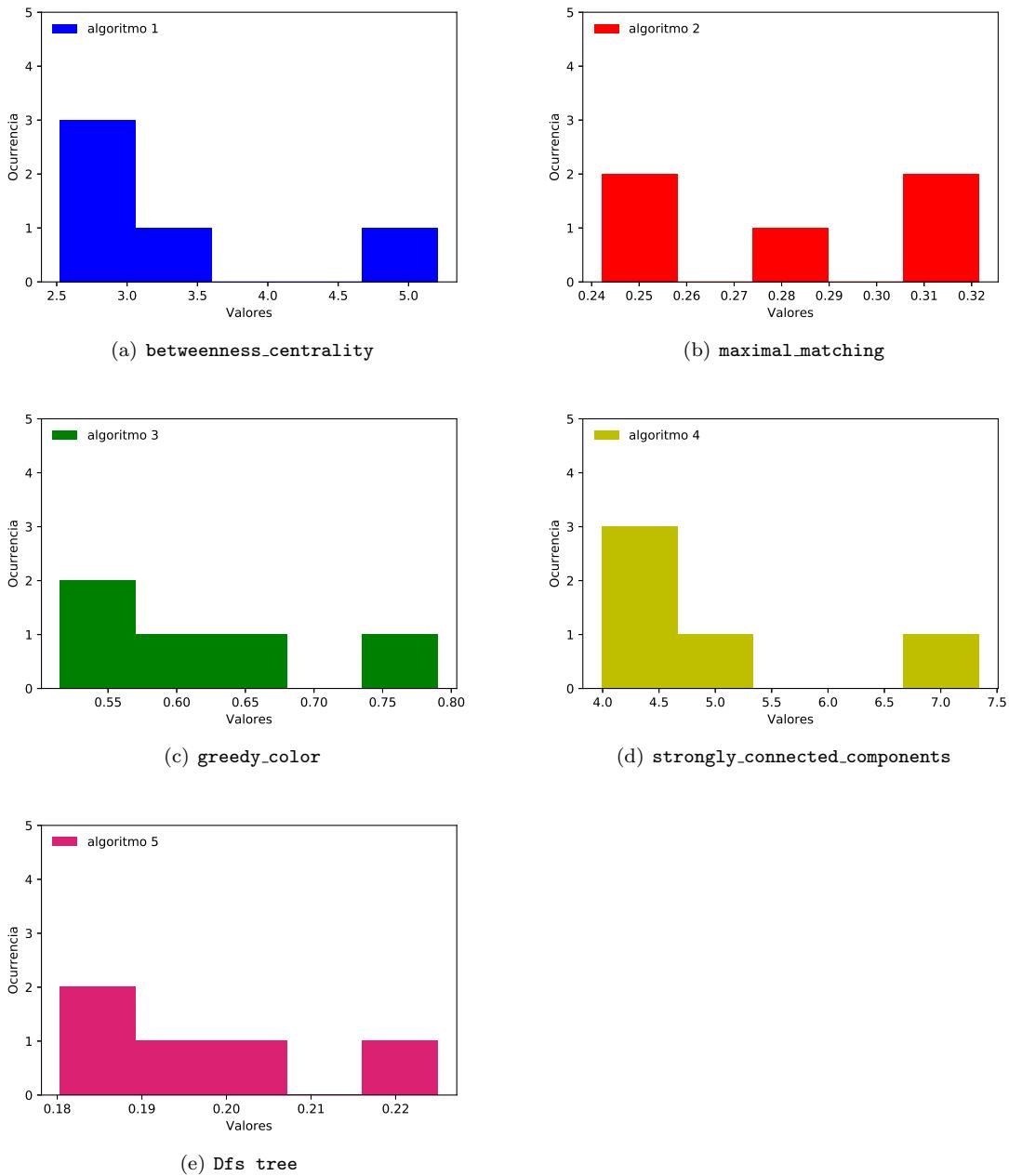


Figura 1: Histograma de cada uno de los cinco algoritmo con los cinco grafos generados.

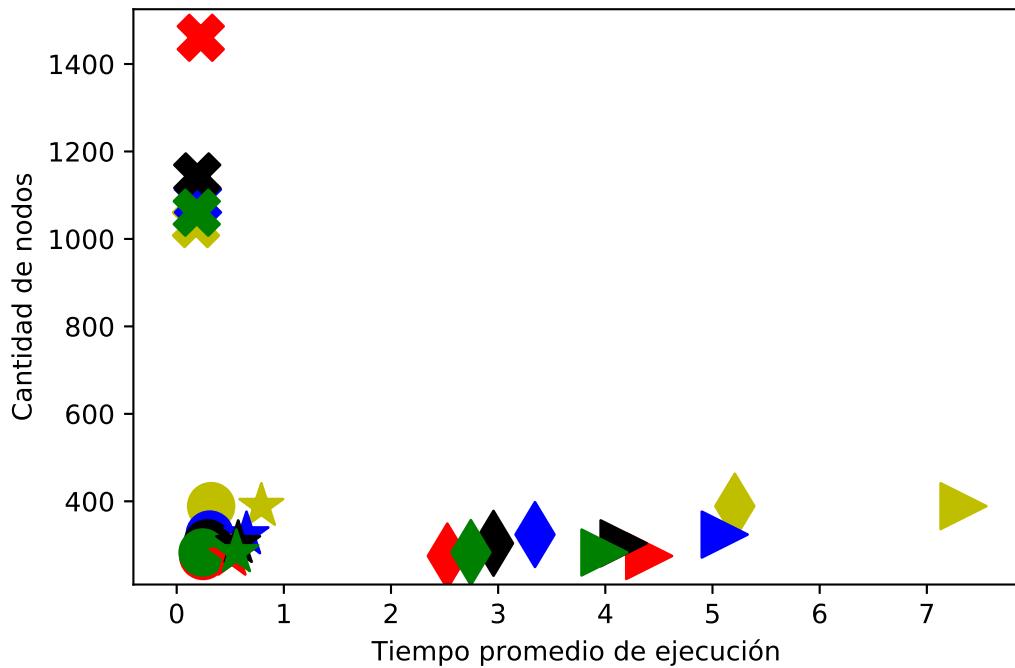


Figura 2: Gráfica de dispersión. Número de nodos respecto al tiempo de ejecución.

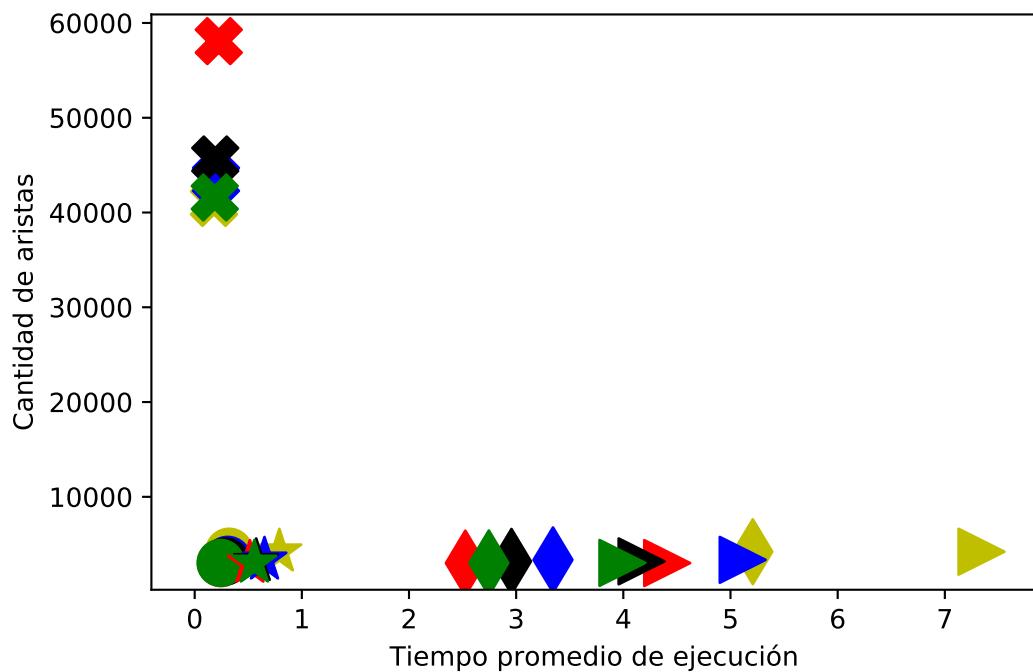


Figura 3: Gráfica de dispersión. Número de aristas respecto al tiempo de ejecución.

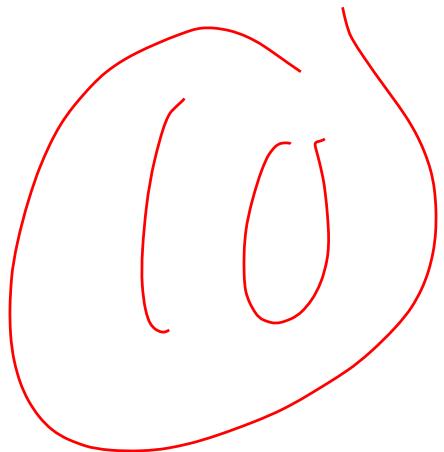
## **Referencias**

- [1] Matplotlib. url`https://matplotlib.org`. Accedido 18-3-2019.
- [2] Networkx. `https://networkx.github.io/documentation/stable/reference/algorithms/`.
- [3] Python data analysis library. url`https://pandas.pydata.org`, 2018). Accedido 18-3-2019.

# Tarea 4

5280

2 de abril de 2019



## Generadores de grafos

En la selección de los algoritmos generadores de grafos de NetworkX [1] se tiene en cuenta la densidad de las aristas que generan.

- `random_tree`: Este algoritmo recibe el número de nodos y devuelve un árbol aleatorio.
- `dense_gnm_random`: Recibe como parámetros directamente el número de nodos y el número de aristas, el algoritmo los distribuye aleatoriamente.
- `erdos_renyi`: Este algoritmo recibe el número de nodos como parámetros y una probabilidad de creación de aristas, por lo tanto la cantidad de aristas varía de un grafo a otro, manteniendo la misma cantidad de nodos.

## Algoritmos de flujo máximo

Los algoritmos de flujo máximo eligen de modo que no generen errores cuando se les pasan grafos con nodos no conexos. Los tres reciben como parámetros el grafo, el nodo fuente y el nodo sumidero.

- `edmonds_karp`: Este algoritmo es una implementación del método de Ford-Fulkerson, con la particularidad de que el orden para ir buscando los caminos está definido.
- `dinitz`: La introducción de los conceptos nivel de grafo y bloqueo de flujo es lo que define el rendimiento de este algoritmo.
- `boykov_kolmogorov`: Aunque este algoritmo está pensado para grafos dirigidos, también funciona con grafos no dirigidos.

## Generación de datos

Son generados diez grafos de 100, 200, 400 y 800 nodos, con cada uno de los tres algoritmos generadores elegidos, empleando pesos con distribución normal alrededor de once con una varianza de siete, como puede observarse en la figura 2.

Los datos para realizar el análisis son guardados en una tabla de 1800 filas que consta de las siguientes columnas.

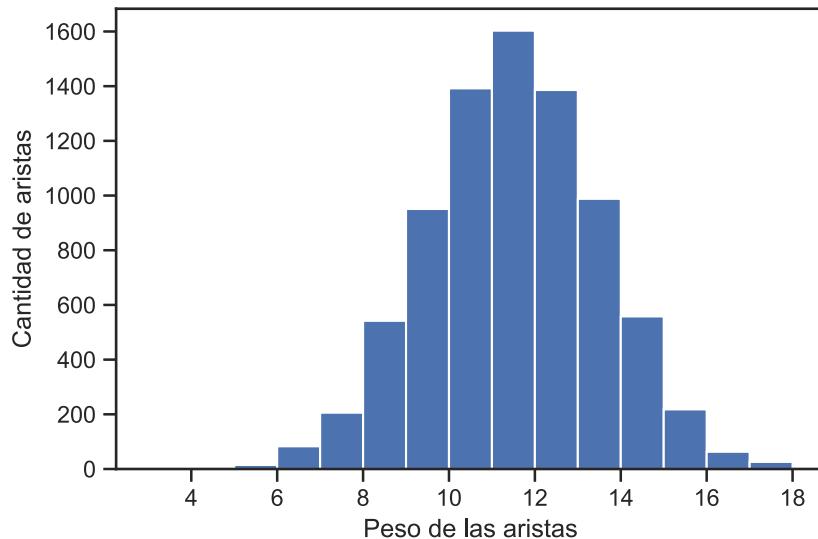


Figura 1: Histograma de los pesos de las aristas de uno de los grafos de 400 nodos generados.

- Generador: Nombre del algoritmo generador de grafos.
- Algoritmo: Nombre del algoritmo de flujo máximo.
- Nodos: Cantidad de nodos.
- Densidad: Densidad del grafo.
- Mediana: Valor de la mediana obtenida de las cinco iteraciones.
- Media: Valor de la media obtenida de las cinco iteraciones.
- Desv: Valor de la desviación estándar obtenida de las cinco *iteraciones*. (circularizado)
- Var: Valor de la varianza obtenida de las cinco iteraciones.

A continuación se muestra el código generador de los datos.

```

1 def GenerateGraphs(size , base):
2     dic=GenerateDic()
3     for i in range(1,5):
4         for j in range(10):
5             G=AddEdges(nx.dense_gnm_random_graph(size ,int(size*size*0.03)))
6             nodes=RandNodes(size-1)
7             for k in range (5):
8                 edmond=[]
9                 din=[]
10                boyk=[]
11                for l in range(5):
12                    edmond.append(Edmond(G, nodes[1][0] , nodes[1][1]))
13                    din.append(Din(G, nodes[1][0] , nodes[1][1]))
14                    boyk.append(Boyk(G, nodes[1][0] , nodes[1][1]))
15                Asign(dic , "dense", "Edmond" , G.number_of_nodes() ,G.number_of_edges()
16 ,nodes[k] ,np.median(edmond) ,np.mean(edmond) ,np.std(edmond) ,np.var(edmond) ,nx.
density(G))
17                Asign(dic , "dense", "Dinitz" , G.number_of_nodes() , G.number_of_edges()
() ,nodes[k] ,np.median(din) ,np.mean(din) ,np.std(din) ,np.var(din) ,nx.density(G))

```

```

17 Asign(dic , "dense" , "Boyk" , G.number_of_nodes() , G.number_of_edges() ,
18 nodes[k] , np.median(boyk) , np.mean(boyk) , np.std(boyk) , np.var(boyk) , nx.density(G) )
19 G=AddEdges(nx.erdos_renyi_graph(size , 0.1))
20 for k in range (5):
21     edmond=[]
22     din=[]
23     boyk=[]
24     for l in range(5):
25         edmond.append(Edmond(G, nodes[1][0] , nodes[1][1]) )
26         din.append(Din(G, nodes[1][0] , nodes[1][1]))
27         boyk.append(Boyk(G, nodes[1][0] , nodes[1][1]))
28     Asign(dic , "erdos" , "Edmond" , G.number_of_nodes() , G.number_of_edges()
29 () , nodes[k] , np.median(edmond) , np.mean(edmond) , np.std(edmond) , np.var(edmond) , nx.
30 density(G))
31     Asign(dic , "erdos" , "Dinitz" , G.number_of_nodes() , G.number_of_edges()
32 () , nodes[k] , np.median(din) , np.mean(din) , np.std(din) , np.var(din) , nx.density(G))
33     Asign(dic , "erdos" , "Boyk" , G.number_of.nodes() , G.number_of.edges()
34 , nodes[k] , np.median(boyk) , np.mean(boyk) , np.std(boyk) , np.var(boyk) , nx.density(G))
35     G=AddEdges(AddEdges(nx.random_tree(size)))
36     for k in range (5):
37         edmond=[ Edmond (G, no ____ ) for l in range(5)]
38         din=[]
39         boyk=[]
40         for l in range(5):
41             edmond.append(Edmond(G, nodes[1][0] , nodes[1][1]) )
42             din.append(Din(G, nodes[1][0] , nodes[1][1]))
43             boyk.append(Boyk(G, nodes[1][0] , nodes[1][1]))
44     Asign(dic , "tree" , "Edmond" , G.number_of.nodes() , G.number_of.edges()
45 , nodes[k] , np.median(edmond) , np.mean(edmond) , np.std(edmond) , np.var(edmond) , nx.
46 density(G))
47     Asign(dic , "tree" , "Dinitz" , G.number_of.nodes() , G.number_of.edges()
48 , nodes[k] , np.median(din) , np.mean(din) , np.std(din) , np.var(din) , nx.density(G))
49     Asign(dic , "tree" , "Boyk" , G.number_of.nodes() , G.number_of.edges()
50 , nodes[k] , np.median(boyk) , np.mean(boyk) , np.std(boyk) , np.var(boyk) , nx.density(G))
51     size*=base;
52 df=pd.DataFrame(dic)
53 df.to_csv("matrix.csv")

```

Generador.py

$$x = [x \neq 2 \text{ for } x \text{ in } range(3)]$$

## Análisis de varianza

$$x = [0, 1, 4]$$

Para el análisis de varianza, en lo adelante ANOVA, por sus siglas en inglés [2], se utilizó el siguiente código.

```

1 import pandas as pd
2 import scipy.stats as stats
3 import matplotlib.pyplot as plt
4 import researchpy as rp
5 import statsmodels.api as sm
6 from statsmodels.formula.api import ols
7 import numpy as np
8 import pingouin as pg
9 import seaborn as sns
10 from statsmodels.stats.multicomp import pairwise_tukeyhsd
11 import csv
12
13 df = pd.read_csv("Matrix.csv", dtype={'Nodos': 'category' , 'Generados': 'category' ,
14 'Densidad': np.float64 })
15 logX = np.log1p(df["Mediana"])
16 df = df.assign(mediana=logX.values)
17 df.drop(["Mediana"] , axis= 1, inplace= True)

```

```

17
18 for i in range(0, df["Densidad"].count()):
19     if df.iat[i, 9] < 0.03716667:
20         df.iat[i, 9] = 1
21     elif df.iat[i, 9] < 0.07183333:
22         df.iat[i, 9] = 2
23     else:
24         df.iat[i, 9] = 3
25 df["Densidad"].replace({1: 'baja', 2: 'media', 3: 'alta'}, inplace= True)
26 factores=[ "Nodos", "Generador", "Algoritmo", "Densidad"]
27 for i in factores:
28     print(rp.summary_cont(df[ 'mediana' ].groupby(df[ i ])))
29     anova = pg.anova (dv='mediana', between=i, data=df, detailed=True , )
30     pg._export_table (anova ,("ANOVA"+i+" .csv"))
31     ax=sns.boxplot(x=df["mediana"], y=df[i], data=df, palette="Set1")
32     plt.savefig(i+" 1.png", bbox_inches='tight')
33     plt.savefig(i + " 1.eps", bbox_inches='tight')
34     tukey = pairwise_tukeyhsd(endog = df["mediana"], groups= df[ i ], alpha=0.05)
35     tukey.plot_simultaneous(xlabel='Time', ylabel=i)
36     plt.vlines(x=49.57,ymin=-0.5,ymax=4.5, color="red")
37     plt.savefig(i+" 2.png", bbox_inches='tight')
38     plt.savefig(i + " 2.eps", bbox_inches='tight')
39     t_csv = open("Tukey"+i+" .csv" , 'w')
40     with t_csv:
41         writer = csv.writer(t_csv)
42         writer.writerow(tukey.summary())
43     plt.show()

```

Anova.py

## Efecto que el generador de grafo usado tiene en el tiempo de ejecución

El ANOVA realizado con los datos obtenidos sobre las mediciones arroja los resultados mostrados en el cuadro 1, o sea, que el generador de grafos sí influye en el tiempo de los algoritmos de flujo.

Cuadro 1

Grupo 1	Grupo 2	Grupo 3	Menos	Mayor	Rechazar
dense	erdos	0.0971	0.0538	0.1405	True
dense	tree	-0.1993	-0.2427	-0.156	True
erdos	tree	-0.2964	-0.3398	-0.2531	True

La figura 1 muestra el diagrama de caja y bigotes para estas mediciones, donde se aprecia que el generador `dense_gnm_random` es el que más influye en el tiempo de ejecución.

## Efecto que el algoritmo de flujo máximo usado tiene en el tiempo de ejecución

El cuadro 2 muestra el resultado ANOVA sobre los algoritmos de flujo máximo. Puede observarse que al comparar `boykov_kolmogorov` y `edmonds_karp`, no existen diferencias entre ellos en cuanto a

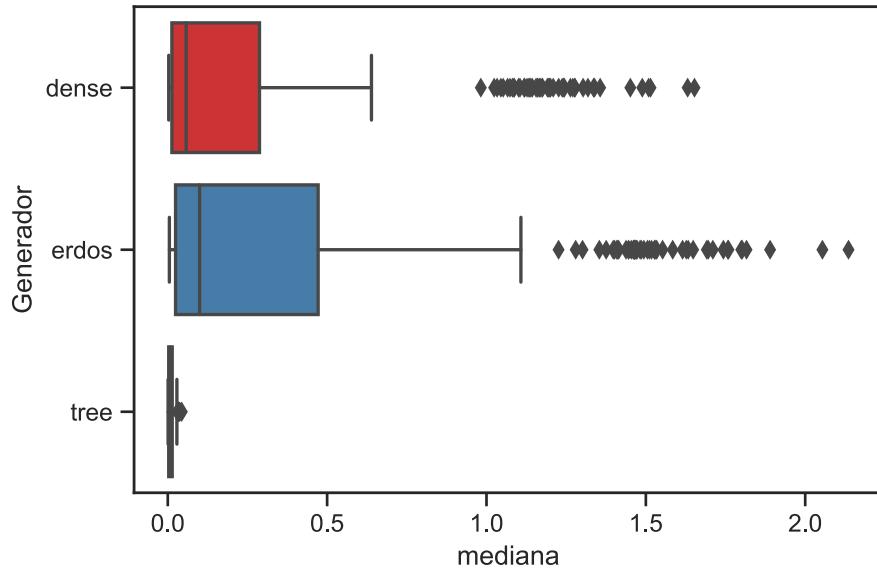


Figura 2: Diagrama de caja y bigotes para los generadores de grafo.

cómo influencian en el tiempo de ejecución, mientras `dinitz` sí tiene un comportamiento diferente a ellos. Esto se observa con mayor claridad en el diagrama de caja y bigotes de la figura 6.

Cuadro 2

Grupo 1	Grupo 2	Grupo 3	Menos	Mayor	Rechazar
Boyk	Dinitz		0.1893	0.1448	0.2338   True
Boyk	Edmond		-0.034	-0.0786	0.0105   False
Dinitz	Edmond		-0.2234	-0.2679	-0.1789   True

## Efecto que el número de nodos del grafo tiene en el tiempo de ejecución

En el cuadro 3 se observa que entre grafos de 100 y 200 nodos no hay un efecto significativo en el tiempo de ejecución. A medida que aumenta el tamaño el efecto va incrementándose. En la figura 3 puede observarse esto gráficamente en el diagrama de caja y bigotes.

## Efecto que la densidad del grafo tiene en el tiempo de ejecución

Para realizar el análisis del efecto de la densidad se realiza un análisis de la distribución de los valores, pues el número de valores de densidad es excesivo para analizarlos en su totalidad. En la figura 7 se muestra el histograma realizado sobre las 1800 mediciones realizadas y puede verse que están distribuidos uniformemente en tres grupos de densidad: baja, media y alta.

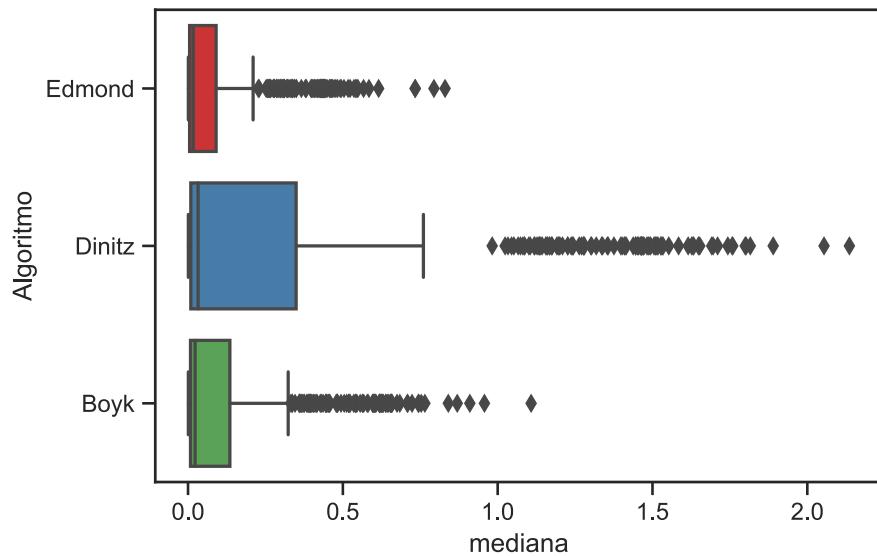


Figura 3: Diagrama de caja y bigotes para los algoritmos de flujo máximo.

Cuadro 3

Grupo 1	Grupo 2	Grupo 3	Menos	Mayor	Rechazar
100	200	0.0216	-0.0252	0.0685	False
100	400	0.1322	0.0853	0.179	True
100	800	0.5155	0.4686	0.5623	True
200	400	0.1105	0.0636	0.1574	True
200	800	0.4938	0.447	0.5407	True
400	800	0.3833	0.3364	0.4302	True

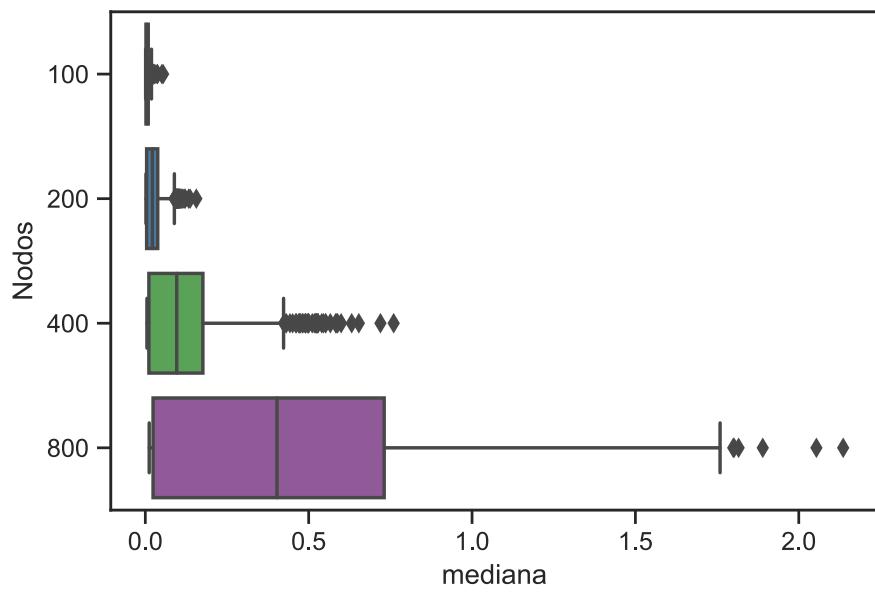


Figura 4: Diagrama de caja y bigotes para el número de nodos de los grafos.

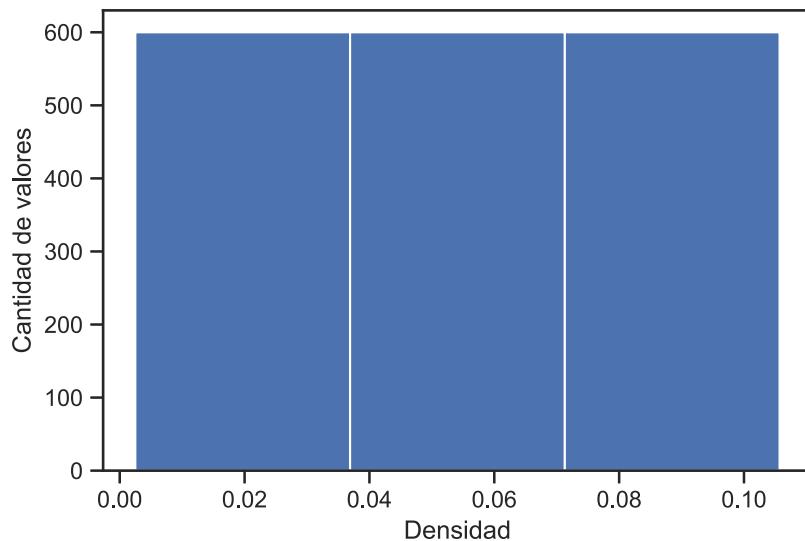


Figura 5: Histograma de los valores de densidad.

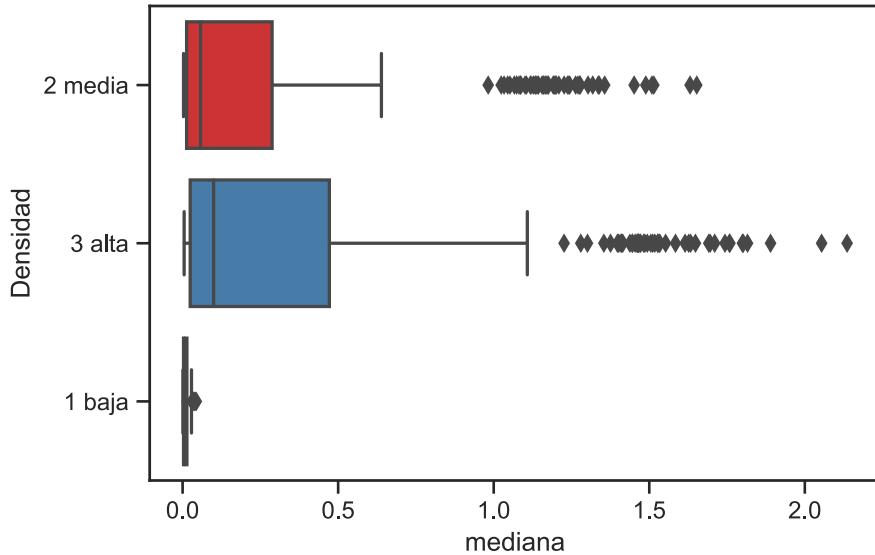


Figura 6: Diagrama de caja y bigotes para la densidad de los grafos.

En el cuadro 4 se muestran los resultados del ANOVA para los valores de densidad y se puede apreciar que sí tiene efecto en el tiempo de ejecución de los algoritmos. Esto se observa de manera gráfica en la figura 8.

Cuadro 4

Grupo 1	Grupo 2	Grupo 3	Menos	Mayor	Rechazar
baja	media	0.1993	0.156	0.2427	True
baja	alta	0.2964	0.2531	0.3398	True
media	alta	0.0971	0.0538	0.1405	True

## Conclusiones

Al realizar un ANOVA sobre los cuatro parámetros, puede apreciarse en el cuadro 5 que la mayor influencia está dada entre los algoritmos de flujo máximo y el número de nodos de los grafos.

Cuadro 5

<i>Factor</i>	sum_sq	df	F	PE(; <i>F</i> )
Generador	-0.009885	2	-0.38830974	1
Algoritmo	-0.08250016	2	-3.66347001	1
Densidad	-0.0052818	2	-0.20762995	1
Generador:Algoritmo	0.0346208	4	0.79983312	0.37126377
Generador:Densidad	0.0346208	4	0.79983323	0.37126374
Algoritmo:Densidad	0.01205205	4	0.38486979	0.67991571
Nodos	1.03E-11	1	8.53E-10	0.99997669
Algoritmo:Nodos	23.596206	2	976.303468	3.16E-287
Nodos:Densidad	0.00972681	2	0.40245243	0.6687357
Generador:Nodos	0.00972681	2	0.37894829	0.68265667
Residual	21.582752	1786		

## Referencias

- [1] Networkx. <https://networkx.github.io/documentation/stable/reference/algorithms/>.
- [2] Python for data science. <https://pythonfordatascience.org/anova-python/>.

# Tarea 4

5280

3 de junio de 2019

## Generadores de grafos

En la selección de los algoritmos generadores de grafos de NetworkX [1] se tiene en cuenta la densidad de las aristas que generan.

- `random_tree`: Este algoritmo recibe el número de nodos y devuelve un árbol aleatorio.
- `dense_gnm_random`: Recibe como parámetros directamente el número de nodos y el número de aristas, el algoritmo los distribuye aleatoriamente.
- `erdos_renyi`: Este algoritmo recibe el número de nodos como parámetros y una probabilidad de creación de aristas, por lo tanto la cantidad de aristas varía de un grafo a otro, manteniendo la misma cantidad de nodos.

## Algoritmos de flujo máximo

Los algoritmos de flujo máximo se eligen de modo que no generen errores cuando se les pasan grafos con nodos no conexos. Los tres reciben como parámetros el grafo, el nodo fuente y el nodo sumidero.

- `edmonds_karp`: Este algoritmo es una implementación del método de Ford-Fulkerson, con la particularidad de que el orden para ir buscando los caminos está definido.
- `dinitz`: La introducción de los conceptos nivel de grafo y bloqueo de flujo es lo que define el rendimiento de este algoritmo.
- `boykov_kolmogorov`: Aunque este algoritmo está pensado para grafos dirigidos, también funciona con grafos no dirigidos.

## Generación de datos

Son generados diez grafos de 100, 200, 400 y 800 nodos, con cada uno de los tres algoritmos generadores elegidos, empleando pesos con distribución normal alrededor de once con una varianza de siete, como puede observarse en la figura 2.

Los datos para realizar el análisis son guardados en una tabla de 1800 filas que consta de las siguientes columnas.

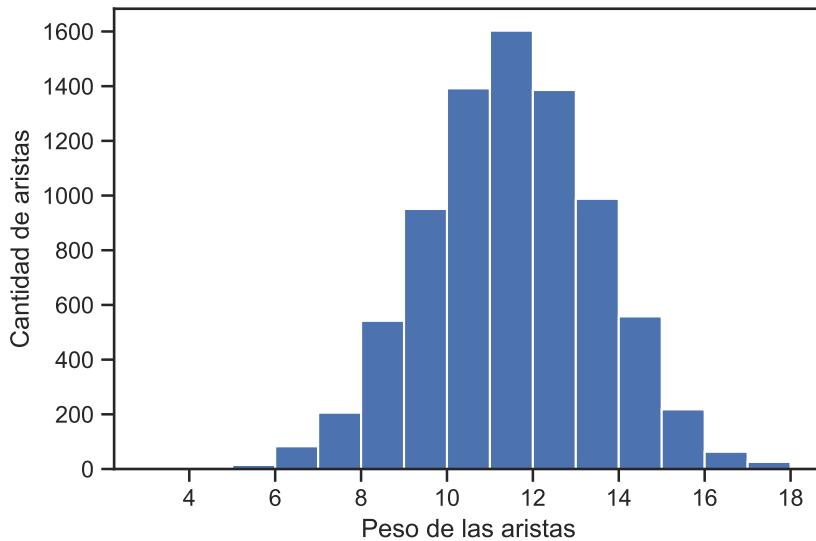


Figura 1: Histograma de los pesos de las aristas de uno de los grafos de 400 nodos generados.

- Generador: Nombre del algoritmo generador de grafos.
- Algoritmo: Nombre del algoritmo de flujo máximo.
- Nodos: Cantidad de nodos.
- Densidad: Densidad del grafo.
- Mediana: Valor de la mediana obtenida de las cinco iteraciones.
- Media: Valor de la media obtenida de las cinco iteraciones.
- Desv: Valor de la desviación estándar obtenida de las cinco iteraciones.
- Var: Valor de la varianza obtenida de las cinco iteraciones.

A continuación se muestra el código generador de los datos.

```

1 def GenerateGraphs( size , base):
2     dic=GenerateDic()
3     for i in range(1,5):
4         for j in range(10):
5             G=AddEdges(nx.dense_gnm_random_graph(size ,int( size *size *0.03 )))
6             nodes=RandNodes(size -1)
7             for k in range (5):
8                 edmond=[]
9                 din []
10                boyk []
11                for l in range(5):
12                    edmond.append(Edmond(G, nodes [1][0] , nodes [1][1]))
13                    din.append(Din(G, nodes [1][0] , nodes [1][1]))
14                    boyk.append(Boyk(G, nodes [1][0] , nodes [1][1]))
15                Asign(dic , "dense" , "Edmond" , G.number_of_nodes() ,G.number_of_edges()
16                ,nodes [k] ,np.median(edmond) ,np.mean(edmond) ,np.std(edmond) ,np.var(edmond) ,nx.
density(G))
17                Asign(dic , "dense" , "Dinitz" , G.number_of_nodes() , G.number_of_edges()
() ,nodes [k] ,np.median(din) ,np.mean(din) ,np.std(din) ,np.var(din) ,nx.density(G))

```

```

17 Asign(dic , "dense" , "Boyk" , G.number_of_nodes() , G.number_of_edges() ,
18 nodes[k] , np.median(boyk) , np.mean(boyk) , np.std(boyk) , np.var(boyk) , nx.density(G))
19 G=AddEdges(nx.erdos_renyi_graph(size , 0.1))
20 for k in range (5):
21     edmond=[]
22     din=[]
23     boyk=[]
24     for l in range(5):
25         edmond.append(Edmond(G, nodes[1][0] , nodes[1][1]) )
26         din.append(Din(G, nodes[1][0] , nodes[1][1]))
27         boyk.append(Boyk(G, nodes[1][0] , nodes[1][1]))
28     Asign(dic , "erdos" , "Edmond" , G.number_of_nodes() , G.number_of_edges()
29 () , nodes[k] , np.median(edmond) , np.mean(edmond) , np.std(edmond) , np.var(edmond) , nx.
30 density(G))
31     Asign(dic , "erdos" , "Dinitz" , G.number_of_nodes() , G.number_of_edges()
32 () , nodes[k] , np.median(din) , np.mean(din) , np.std(din) , np.var(din) , nx.density(G))
33     Asign(dic , "erdos" , "Boyk" , G.number_of.nodes() , G.number_of.edges()
34 , nodes[k] , np.median(boyk) , np.mean(boyk) , np.std(boyk) , np.var(boyk) , nx.density(G))
35     G=AddEdges(AddEdges(nx.random_tree(size)))
36     for k in range (5):
37         edmond=[]
38         din=[]
39         boyk=[]
40         for l in range(5):
41             edmond.append(Edmond(G, nodes[1][0] , nodes[1][1]) )
42             din.append(Din(G, nodes[1][0] , nodes[1][1]))
43             boyk.append(Boyk(G, nodes[1][0] , nodes[1][1]))
44     Asign(dic , "tree" , "Edmond" , G.number_of.nodes() , G.number_of.edges()
45 , nodes[k] , np.median(edmond) , np.mean(edmond) , np.std(edmond) , np.var(edmond) , nx.
46 density(G))
47     Asign(dic , "tree" , "Dinitz" , G.number_of.nodes() , G.number_of.edges()
48 , nodes[k] , np.median(din) , np.mean(din) , np.std(din) , np.var(din) , nx.density(G))
49     Asign(dic , "tree" , "Boyk" , G.number_of.nodes() , G.number_of.edges()
50 , nodes[k] , np.median(boyk) , np.mean(boyk) , np.std(boyk) , np.var(boyk) , nx.density(G))
51     size*=base;
52 df=pd.DataFrame(dic)
53 df.to_csv("matrix.csv")

```

Generador.py

## Análisis de varianza

Para el análisis de varianza, en lo adelante ANOVA, por sus siglas en inglés [2], se utilizó el siguiente código.

```

1 import pandas as pd
2 import scipy.stats as stats
3 import matplotlib.pyplot as plt
4 import researchpy as rp
5 import statsmodels.api as sm
6 from statsmodels.formula.api import ols
7 import numpy as np
8 import pingouin as pg
9 import seaborn as sns
10 from statsmodels.stats.multicomp import pairwise_tukeyhsd
11 import csv
12
13 df = pd.read_csv("Matrix.csv" , dtype={ 'Nodos': 'category' , 'Generados': 'category' ,
14 'Densidad': np.float64 })
15 logX = np.log1p(df["Mediana"])
16 df = df.assign(mediana=logX.values)
17 df.drop(["Mediana"] , axis= 1 , inplace= True)

```

```

17
18 for i in range(0, df["Densidad"].count()):
19     if df.iat[i, 9] < 0.03716667:
20         df.iat[i, 9] = 1
21     elif df.iat[i, 9] < 0.07183333:
22         df.iat[i, 9] = 2
23     else:
24         df.iat[i, 9] = 3
25 df["Densidad"].replace({1: 'baja', 2: 'media', 3: 'alta'}, inplace= True)
26 factores=[ "Nodos", "Generador", "Algoritmo", "Densidad"]
27 for i in factores:
28     print(rp.summary_cont(df[ 'mediana' ].groupby(df[ i ])))
29     anova = pg.anova (dv='mediana', between=i, data=df, detailed=True , )
30     pg._export_table (anova ,("ANOVA"+i+" .csv"))
31     ax=sns.boxplot(x=df["mediana"], y=df[i], data=df, palette="Set1")
32     plt.savefig(i+" 1.png", bbox_inches='tight')
33     plt.savefig(i+" 1.eps", bbox_inches='tight')
34     tukey = pairwise_tukeyhsd(endog = df["mediana"], groups= df[ i ], alpha=0.05)
35     tukey.plot_simultaneous(xlabel='Time', ylabel=i)
36     plt.vlines(x=49.57,ymin=-0.5,ymax=4.5, color="red")
37     plt.savefig(i+" 2.png", bbox_inches='tight')
38     plt.savefig(i+" 2.eps", bbox_inches='tight')
39     t_csv = open("Tukey"+i+" .csv" , 'w')
40     with t_csv:
41         writer = csv.writer(t_csv)
42         writer.writerow(tukey.summary())
43     plt.show()

```

Anova.py

## Efecto que el generador de grafo usado tiene en el tiempo de ejecución

El ANOVA realizado con los datos obtenidos sobre las mediciones arroja los resultados mostrados en el cuadro 1, o sea, que el generador de grafos sí influye en el tiempo de los algoritmos de flujo máximo.

Cuadro 1: Influencia de generador de grafos en el tiempo de ejecución de los algoritmos de flujo máximo

Grupo 1	Grupo 2	Grupo 3	Menos	Mayor	Rechazar
dense	erdos		0.0971	0.0538	0.1405
dense	tree		-0.1993	-0.2427	-0.156
erdos	tree		-0.2964	-0.3398	-0.2531

La figura 1 muestra el diagrama de caja y bigotes para estas mediciones, donde se aprecia que el generador `dense_gnm_random` es el que más influye en el tiempo de ejecución.

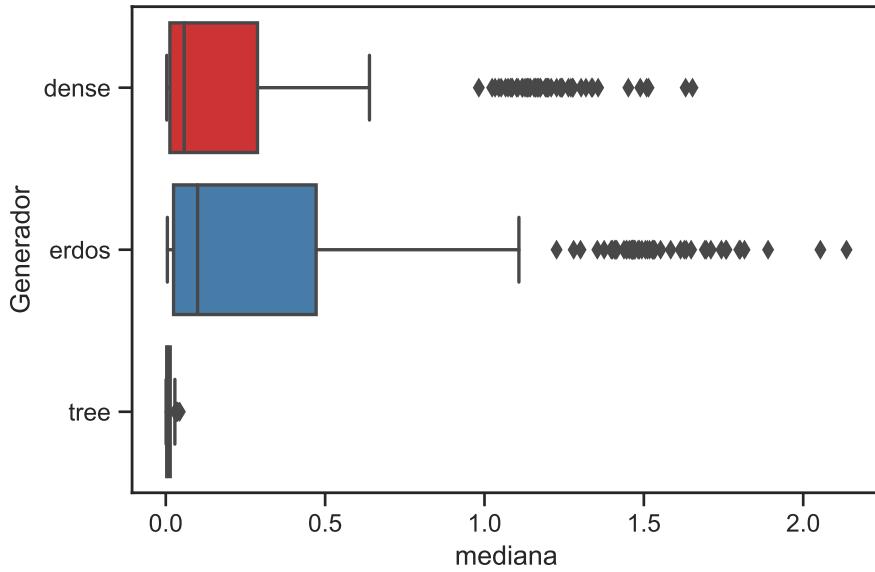


Figura 2: Diagrama de caja y bigotes para los generadores de grafo.

## Efecto que el algoritmo de flujo máximo usado tiene en el tiempo de ejecución

El cuadro 2 muestra el resultado ANOVA sobre los algoritmos de flujo máximo. Puede observarse que al comparar `boykov_kolmogorov` y `edmonds_karp`, no existen diferencias entre ellos en cuanto a cómo influencian en el tiempo de ejecución, mientras `dinitz` sí tiene un comportamiento diferente a ellos. Esto se observa con mayor claridad en el diagrama de caja y bigotes de la figura 6.

Cuadro 2: ANOVA sobre los algoritmos de flujo máximo

Grupo 1	Grupo 2	Grupo 3	Menos	Mayor	Rechazar
Boyk	Dinitz		0.1893	0.1448	0.2338   Verdadero
Boyk	Edmond		-0.034	-0.0786	0.0105   Falso
Dinitz	Edmond		-0.2234	-0.2679	-0.1789   Verdadero

## Efecto que el número de nodos del grafo tiene en el tiempo de ejecución

En el cuadro 3 se observa que entre grafos de 100 y 200 nodos no hay un efecto significativo en el tiempo de ejecución. A medida que aumenta el tamaño el efecto va incrementándose. En la figura 3 puede observarse esto gráficamente en el diagrama de caja y bigotes.

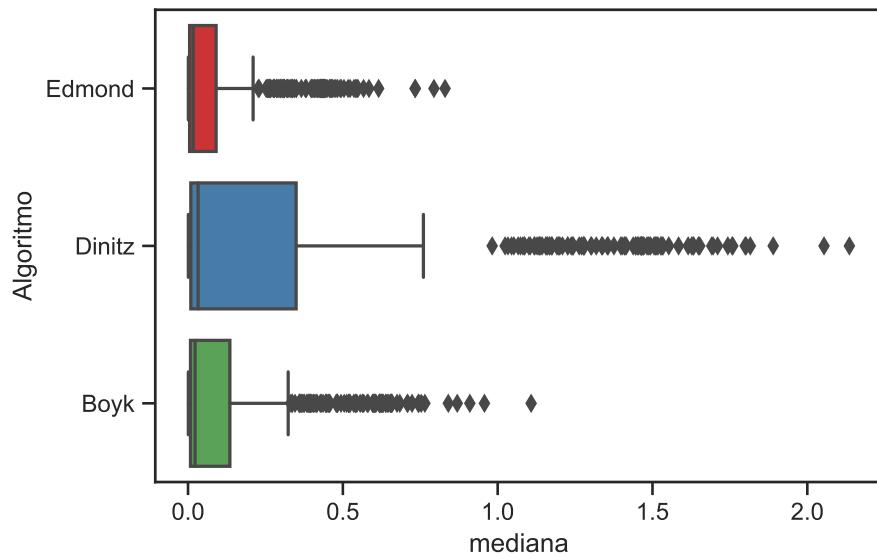


Figura 3: Diagrama de caja y bigotes para los algoritmos de flujo máximo.

Cuadro 3: ANOVA sobre el tiempo de ejecución

Grupo 1	Grupo 2	Grupo 3	Menos	Mayor	Rechazar
100	200	0.0216	-0.0252	0.0685	Falso
100	400	0.1322	0.0853	0.179	Verdadero
100	800	0.5155	0.4686	0.5623	Verdadero
200	400	0.1105	0.0636	0.1574	Verdadero
200	800	0.4938	0.447	0.5407	Verdadero
400	800	0.3833	0.3364	0.4302	Verdadero

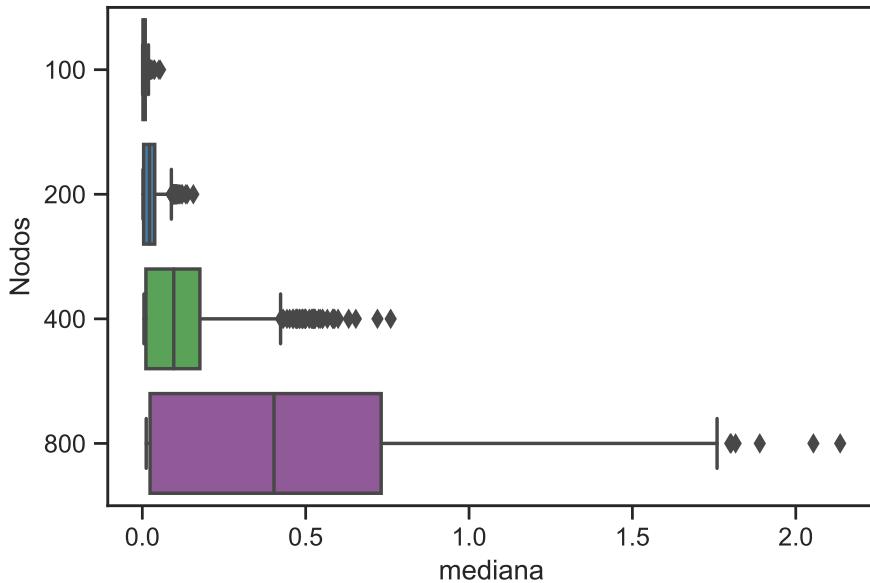


Figura 4: Diagrama de caja y bigotes para el número de nodos de los grafos.

## Efecto que la densidad del grafo tiene en el tiempo de ejecución

Para realizar el análisis del efecto de la densidad se realiza un análisis de la distribución de los valores, pues el número de valores de densidad es excesivo para analizarlos en su totalidad. En la figura 8 se muestra el histograma realizado sobre las 1800 mediciones realizadas y puede verse que están distribuidos uniformemente en tres grupos de densidad: baja, media y alta.

En el cuadro 4 se muestran los resultados del ANOVA para los valores de densidad y se puede apreciar que sí tiene efecto en el tiempo de ejecución de los algoritmos. Esto se observa de manera gráfica en la figura 8.

Cuadro 4: ANOVA sobre los valores de densidad

Grupo 1	Grupo 2	Grupo 3	Menos	Mayor	Rechazar
baja	media	0.1993	0.156	0.2427	Verdadero
baja	alta	0.2964	0.2531	0.3398	Verdadero
media	alta	0.0971	0.0538	0.1405	Verdadero

## Conclusiones

Al realizar un ANOVA sobre los cuatro parámetros, puede apreciarse en el cuadro 5 que la mayor influencia está dada entre los algoritmos de flujo máximo y el número de nodos de los grafos.

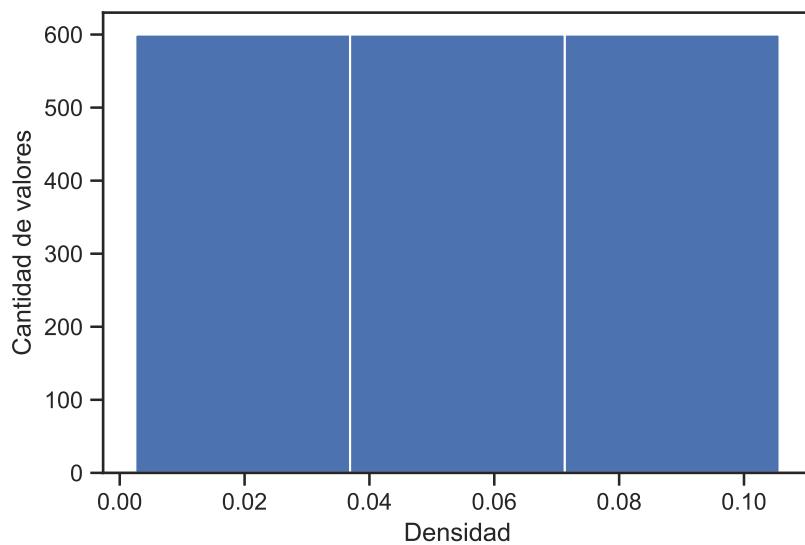


Figura 5: Histograma de los valores de densidad.

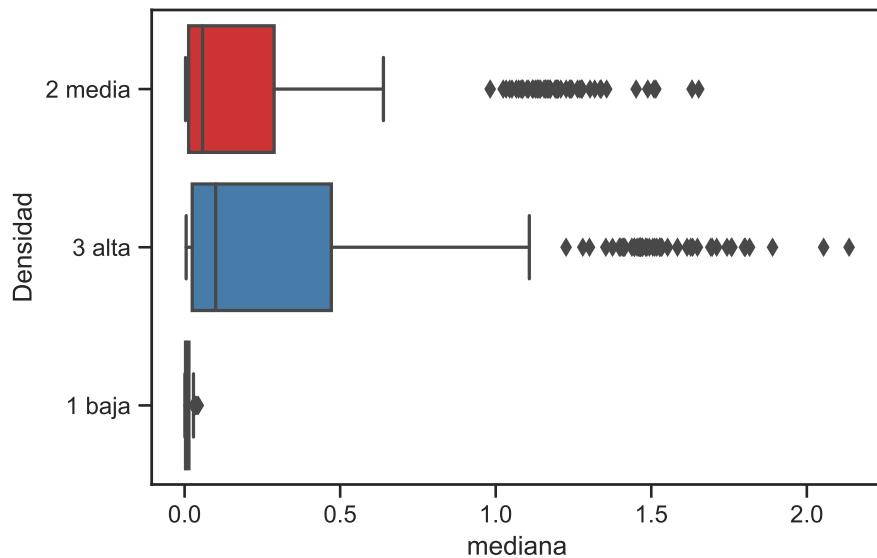


Figura 6: Diagrama de caja y bigotes para la densidad de los grafos.

Cuadro 5: ANOVA sobre los cuatro parámetros

Factor	suma_cuad	Grados de libertad	F	Prueba)
<b>Generador</b>	-0.009	2	-0.9	1
<b>Algoritmo</b>	-0.08	2	-3.67	1
<b>Densidad</b>	-0.008	2	-0.21	1
<b>Generador:Algoritmo</b>	0.03	4	0.79	0.37
<b>Generador:Densidad</b>	0.03	4	0.79	0.37
<b>Algoritmo:Densidad</b>	0.01	4	0.38	0.67
<b>Nodos</b>	0	1	0	0.99
<b>Algoritmo:Nodos</b>	23.59	2	976.3	0
<b>Nodos:Densidad</b>	0.009	2	0.40	0.66
<b>Generador:Nodos</b>	0.009	2	0.37	0.68
<b>Residual</b>	21.58	1786		

## Referencias

- [1] Networkx. <https://networkx.github.io/documentation/stable/reference/algorithms/>.
- [2] Python for data science. <https://pythonfordatascience.org/anova-python/>.

# Tarea 5

5280

30 de abril de 2019



## Generador de grafos

En los grafos presentados, los nodos representan enruteadores en redes locales de comunicación. Se asume que la capacidad de los enlaces está determinada por el tipo de enlace físico que une los enruteadores y no por las características de los mismos.

Para la generación se emplea el algoritmo `dense_gnm_random`, que recibe como parámetros directamente el número de nodos y el número de aristas, lo que resulta cómodo para simular este tipo de redes.

## Algoritmo de flujo máximo

El algoritmo de flujo máximo elegido fue el Edmond Karp, pues fue el que mejores resultados mostró en la tarea anterior. Este algoritmo recibe como parámetros el grafo, el nodo fuente y el nodo sumidero.

## Algoritmo de ordenamiento

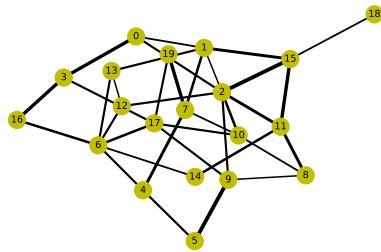
El algoritmo de ordenamiento empleado es el propuesto por Fruchterman y Reingold [1]. Este muestra buenos resultados en la distribución de nodos, tamaño de las aristas uniforme y simetría de manera general, enfocándose en la velocidad y simplicidad.

## Generación de grafos

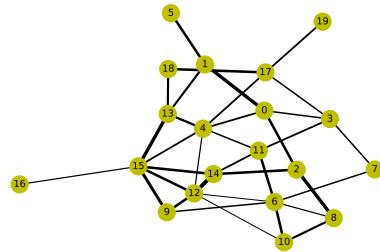
Para realizar el experimento son generados 5 grafos aleatoriamente, con la capacidad de flujo en sus aristas, distribuida normalmente, como puede observarse en la figura 1. El ancho de las aristas es proporcional a su capacidad.

## Características estructurales de los nodos

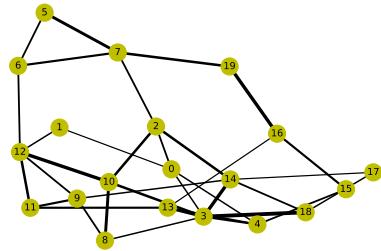
A cada uno de los grafos se les calculan las siguientes características estructurales de sus nodos:



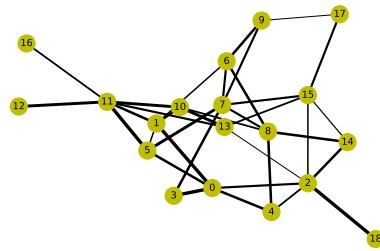
(a) Grafo 1



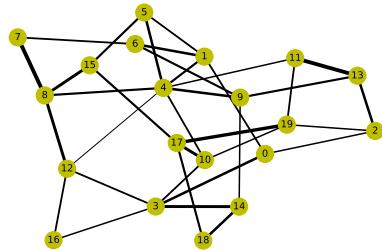
(b) Grafo 2



(c) Grafo 3



(d) Grafo 4



(e) Grafo 5

Figura 1: Grafos generados

- Distribución de grado (figura 2)
- Coeficiente de agrupamiento (figura 3)
- Centralidad de cercanía (figura 4)
- Centralidad de carga (figura 5)
- Excentricidad (figura 6)
- Rango de página (figura 7)

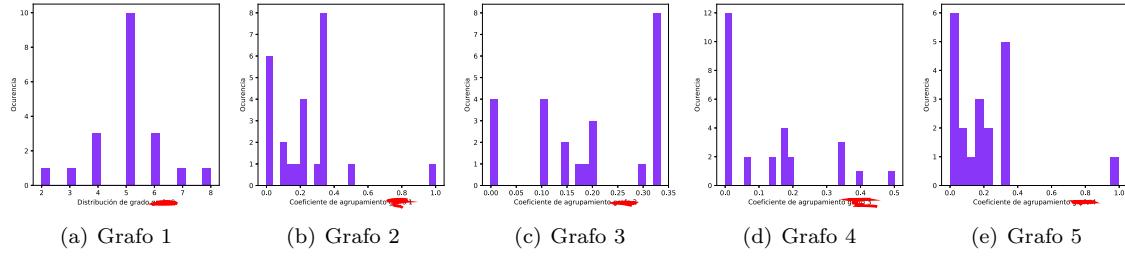


Figura 2: Histogramas de la distribución de grado para cada grafo

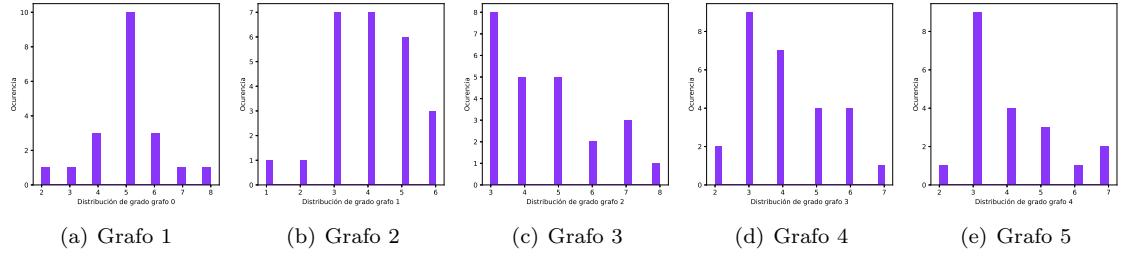


Figura 3: Histogramas del coeficiente de agrupamiento para cada grafo

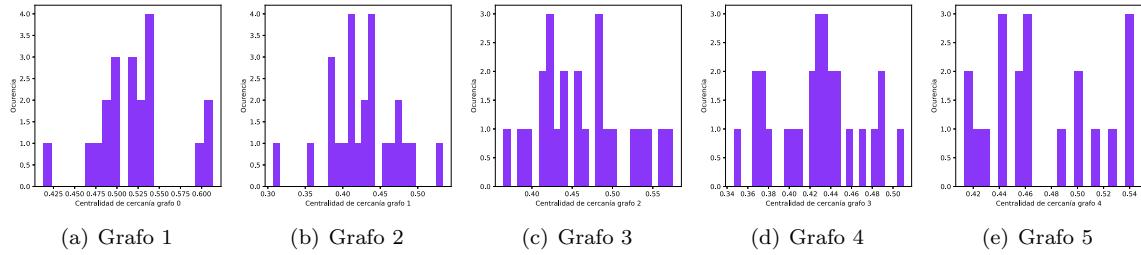


Figura 4: Histogramas de la centralidad de cercanía para cada grafo

## Flujo máximo

Para cada combinación de fuente-sumidero en cada grafo se calcula el flujo máximo. En las figuras 8, 9, 10, 11 y 12, se visualizan los flujos para la mejor y peor pareja de fuente-sumidero para los grafos 1, 2, 3, 4 y 5, respectivamente. Los nodos rojos representan las mejores fuente-sumidero, como los

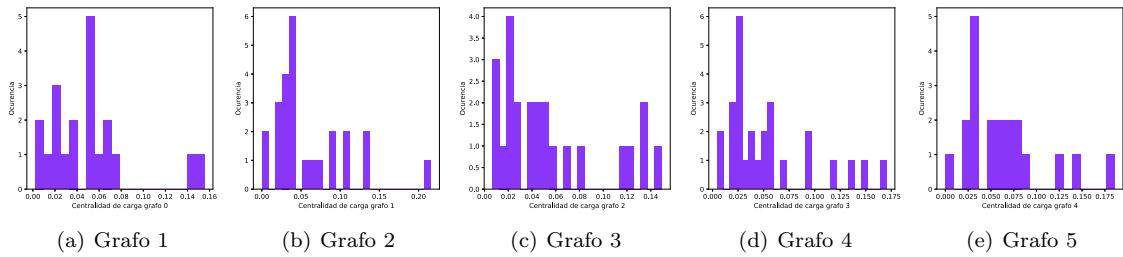


Figura 5: Histogramas de la centralidad de carga para cada grafo

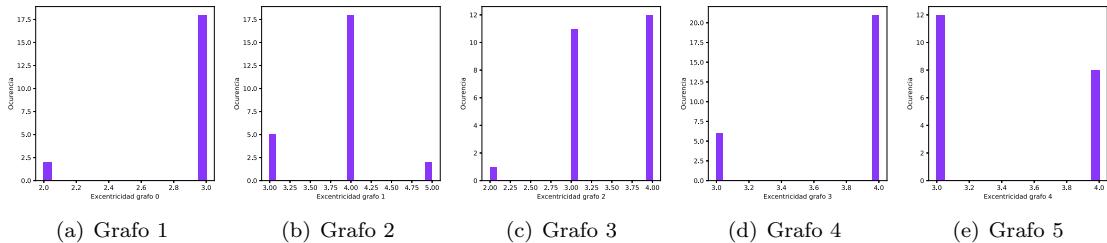


Figura 6: Histogramas de la excentricidad para cada grafo

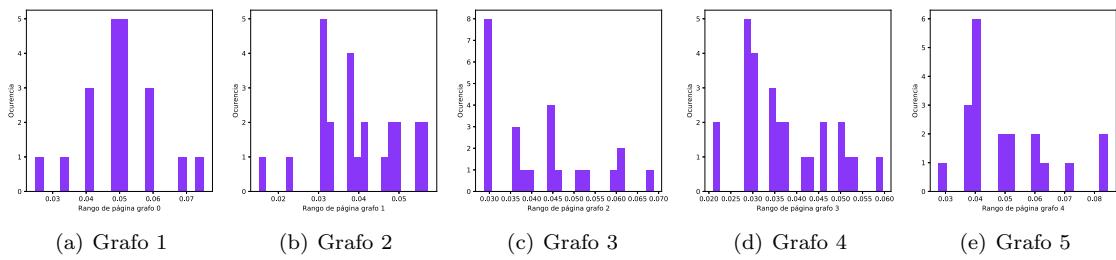
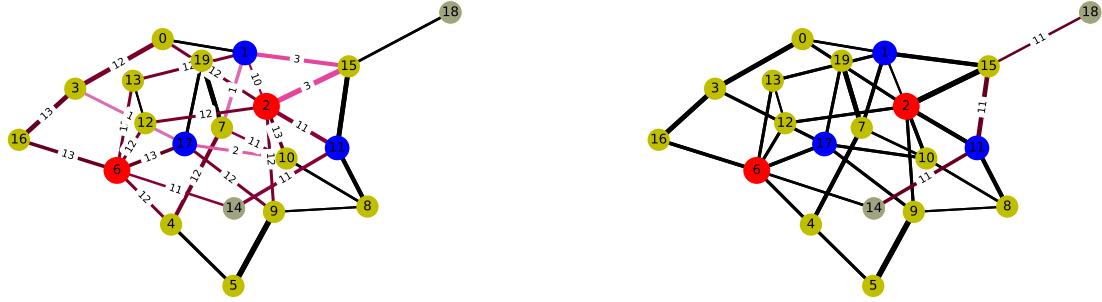


Figura 7: Histogramas de la excentricidad para cada grafo

grafos son no dirigidos, estos son intercambiables. Los nodos azules son buenas fuentes-sumidero y los nodos grises son las peores fuentes-sumideros, en cada grafo.

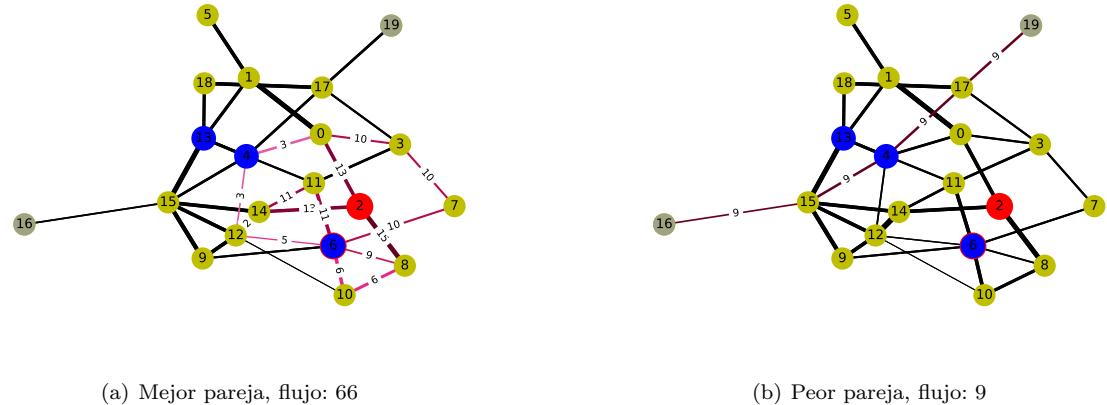
En rosado se muestra cuánto flujo pasa por cada arista una vez aplicado al algoritmo de flujo máximo, aumentando la intensidad del color proporcionalmente a la cantidad de flujo. En las aristas negras no pasa flujo.



(a) Mejor pareja, flujo: 73

(b) Peor pareja, flujo: 11

Figura 8: Visualización de la mejor y peor pareja de nodos a usar como fuente o sumidero en el grafo 1



(a) Mejor pareja, flujo: 66

(b) Peor pareja, flujo: 9

Figura 9: Visualización de la mejor y peor pareja de nodos a usar como fuente o sumidero en el grafo 2

## Influencia de las características estructurales ~~del grafo en el tiempo de ejecución~~

En el Cuadro 1 se muestran los resultados del análisis de varianza (ANOVA en lo adelante, por sus siglas en inglés) de las características estructurales del grafo para ver su influencia en el tiempo de ejecución del algoritmo de flujo máximo.

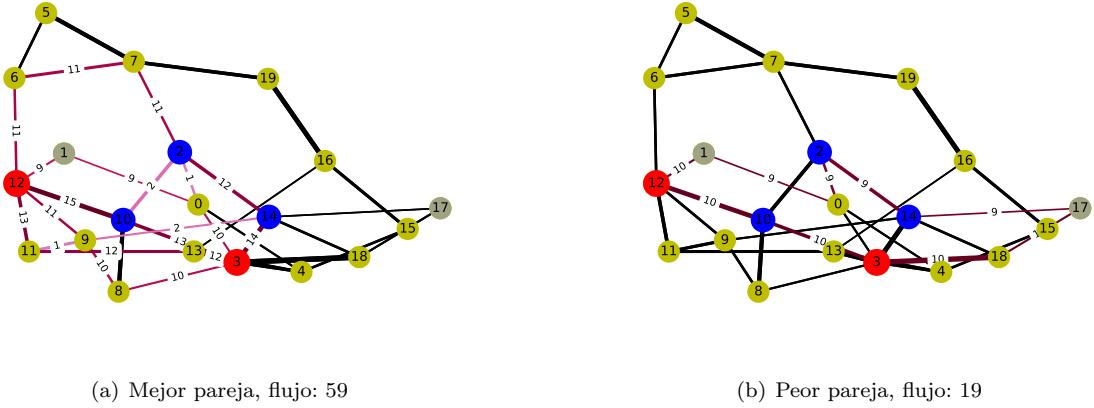


Figura 10: Visualización de la mejor y peor pareja de nodos a usar como fuente o sumidero en el grafo 3

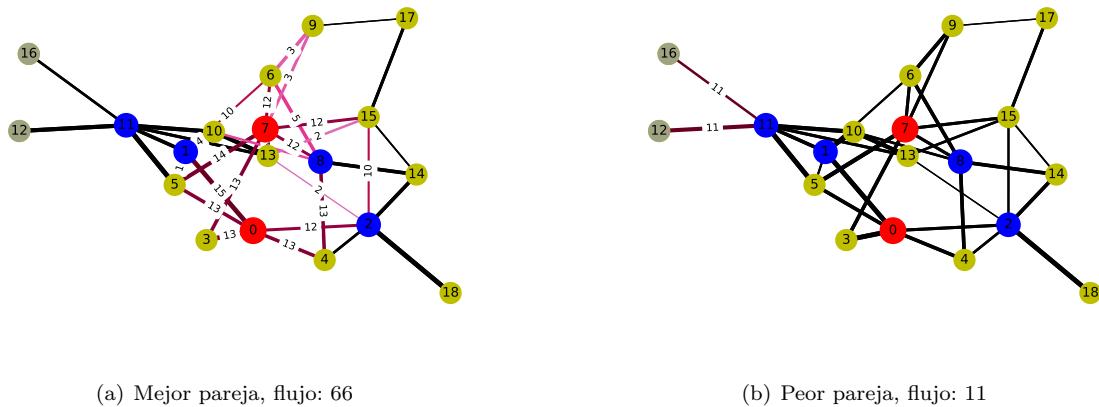
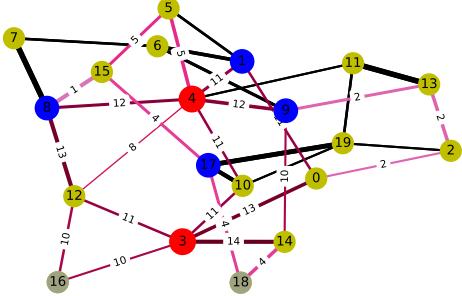
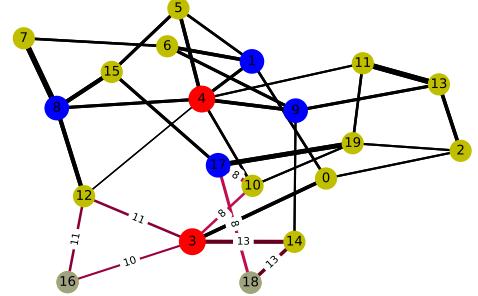


Figura 11: Visualización de la mejor y peor pareja de nodos a usar como fuente o sumidero en el grafo 4



(a) Mejor pareja, flujo: 59



(b) Peor pareja, flujo: 21

Figura 12: Visualización de la mejor y peor pareja de nodos a usar como fuente o sumidero en el grafo 5.

Se aprecia que las características estructurales de los grafos no influyen en el tiempo de ejecución del algoritmo de flujo máximo para los grafos generados. Se observa, además, que existe una relación entre las siguientes características:

- Distribución de grado y coeficiente de agrupamiento
- Distribución de grado y centralidad de cercanía
- Distribución de grado y centralidad de carga
- Rango de página y centralidad de cercanía
- Rango de página y centralidad de carga

Cuadro 1: Resultado del ANOVA para los valores de tiempo

Factor	suma_cuad	Grados de libertad	F	Prueba
Grado	8.12E-08	1	0.96	0.33
CoefA	6.78E-08	1	0.8	0.37
CentCe	5.12E-09	1	0.06	0.8
CentCa	2.16E-07	1	2.5	0.11
Excent	2.32E-08	1	0.27	0.6
RangoP	2.11E-07	1	2.48	0.11
Grado:CoefA	4.45E-07	1	5.23	0.02
Grado:CentCe	1.00E-06	1	11.8	0.0006
Grado:CentCa	1.55E-06	1	18.28	2.00E-05
Grado:Excent	1.84E-07	1	2.16	0.14
Grado:RangoP	2.30E-07	1	2.7	0.1
CentCe:CentCa	2.11E-07	1	2.48	0.12
CentCe:Excent	1.53E-07	1	1.8	0.18
CentCe:RangoP	7.70E-07	1	9.06	0.002
CentCa:Excent	1.30E-07	1	1.53	0.22
CentCa:RangoP	1.89E-06	1	22.25	2.50E-06
Excent:RangoP	1.52E-07	1	1.79	0.18
Residual	0.00015678	1844		

$2.11 \times 10^{-7}$

## Influencia de las características estructurales del grafo en el flujo máximo

En el Cuadro 1 se muestra el ANOVA de las características estructurales del grafo para analizar su influencia en el flujo máximo.

Se aprecia inciden en el flujo máximo las siguientes características estructurales:

- Coeficiente de agrupamiento
- Centralidad de carga
- Rango de página

Se observa, además, que existe una relación entre las siguientes características:

- Distribución de grado y coeficiente de agrupamiento
- Distribución de grado y rango de página
- Coeficiente de agrupamiento y centralidad de cercanía

- Coeficiente de agrupamiento y rango de página

Cuadro 2: Resultado del ANOVA para los valores de flujo máximo

Factor	sum_cuad	Grados de libertad	F	Prueba
Grado	3857288463	1	0.459047	0.498155
CoefA	136323246	1	16.15817	6.06E-05
CentCe	143623743	1	1.702349	0.192144
CentCa	976245999	1	11.57129	0.000684
Excent	155426353	1	1.842243	0.174855
PageR	508236449	1	6.024044	0.014204
Grado:CoefA	650232041	1	7.707095	0.005556
Grado:CentCe	93.006395	1	1.10239	0.29388
Grado:CentCa	112.234812	1	1.330301	0.248901
Grado:Excent	162.655867	1	1.927934	0.165153
Grado:PageR	660247786	1	7.82581	0.005204
CoefA:CentCe	335675561	1	3.978708	0.046226
CoefA:CentCa	49.9598544	1	0.592166	0.441682
CoefA:Excent	16.9158218	1	0.2005	0.65437
CoefA:PageR	553.464369	1	6.560123	0.010508
CentCe:CentCa	7.00601142	1	0.083041	0.77325
CentCe:Excent	23.2094891	1	0.275098	0.599995
CentCe:PageR	85.7199062	1	1.016024	0.313596
CentCa:Excent	66.437239	1	0.78747	0.374982
CentCa:PageR	106.077006	1	1.257314	0.262307
Excent:PageR	104.744206	1	1.241516	0.265325
Residual	155237.086	1840		

## Código fuente utilizado para realizar el experimento

### Generación de grafos

```

1 def GenerateGraph(nodes ,edges ,address):
2     S=nx.dense_gnm_random_graph(nodes ,edges)
3     scale = 2
4     rang = 10
5     e=S.edges(nbunch=None, data=True, default=None)
6     X = truncnorm(a=-rang/scale , b=+rang/scale , scale=scale).rvs(size=edges)
7     X = X.round().astype(int)+rang+2
8     G=nx.Graph()
9     count=0;
10    for i in e:

```

```

11     G.add_edge(i[0], i[1], capacity=X[count])
12     count+=1
13 df = pd.DataFrame()
14 df = nx.to_pandas_adjacency(G, dtype=int, weight='capacity')
15 df.to_csv(address, index=None, header=None)
16
17 def Print(graph, address, pos_address, fig):
18     ds = pd.read_csv(graph, header=None)
19     G = nx.from_pandas_adjacency(ds)
20     pos=nx.fruchterman_reingold_layout(G)
21     X=[]
22     for edge in G.edges():
23         X.append( G.edges[edge]['weight'] )
24     X[:] = [x/10*x/8 for x in X]
25     labels = {}
26     for i in G.nodes:
27         labels[i]=str(i)
28     nx.draw_networkx_nodes(G, pos, node_size=200, node_color='y', node_shape='o')
29     nx.draw_networkx_edges(G, pos, edge_color='black', width=X)
30     nx.draw_networkx_labels(G, pos, labels, font_size=8)
31     plt.axis('off')
32     plt.savefig(fig, dpi=500)
33     df = pd.DataFrame(pos)
34     df.to_csv(address, index=None, header=None)
35     return(G)

```

Grafos.py

## Cálculo de atributos

```

1 def Atributes(G):
2     dic={}
3     Nodes=G.nodes;
4     dic["Nodo"] = Nodes
5     dic["Grado"] =[G.degree(i) for i in Nodes]
6     dic["CoefA"] =[nx.clustering(G,i) for i in Nodes]
7     dic["CentCe"] =[nx.closeness_centrality(G,i) for i in Nodes]
8     dic["CentCa"] =[nx.load_centrality(G,i) for i in Nodes]
9     dic["Excent"] =[nx.eccentricity(G,i) for i in Nodes]
10    PageR=nx.pagerank(G,weight="capacity")
11    dic["PageR"]=[PageR[i] for i in Nodes]
12    df=pd.DataFrame(dic)
13    df.to_csv("matrix5.csv", index=None)

```

Programa.py

## Cálculo de tiempo de ejecución y flujo máximo

```

1 def Time(G):
2     dic={"Fuente":[] , "Sumidero":[] , "Media":[] , "Mediana":[] , "Std":[] , "MaxFlow":[]}
3     Nodes=G.nodes;
4     for i in Nodes:
5         for j in Nodes:
6             if i!=j:
7                 t=[]
8                 for k in range(10):
9                     t.append(Edmond(G,i,j))
10                dic["Fuente"].append(i)
11                dic["Sumidero"].append(j)

```

```

12     dic[ "Media" ].append( np.mean(t) )
13     dic[ "Mediana" ].append( np.median(t) )
14     dic[ "Std" ].append( np.std(t) )
15     dic[ "MaxFlow" ].append( nx.maximum_flow_value(G, i ,j , capacity=" weight" ) )
16   )
17 df=pd.DataFrame( dic )
df.to_csv( "times5.csv" , index=None)

```

Programa.py

## ANOVA

```

1 modelo = ols('Flujomaximo ~ Grado+CoefA+CentCe+CentCa+Excent+PageR+Grado*CoefA+Grado
               *CentCe+Grado*CentCa+Grado*Excent+Grado*PageR+CentCe*CentCa+CentCe*Excent+CentCe
               *PageR+CentCa*Excent+CentCa*PageR+Excent*PageR' , data=df). fit()
2 print( modelo.summary() )
3 modelo_csv = open("Anova_Mult.csv" , 'w')
4 aov_table = sm.stats.anova_lm(modelo , typ=2)
5 df1=pd.DataFrame(aov_table)

```

procesamiento.py

## Referencias

- [1] Thomas MJ Fruchterman and Edward M Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.

# Tarea 5

5280

3 de junio de 2019

## Generador de grafos

En los grafos presentados, los nodos representan enrutadores en redes locales de comunicación. Se asume que la capacidad de los enlaces está determinada por el tipo de enlace físico que une los enrutadores y no por las características de los mismos.

Para la generación se emplea el algoritmo `dense_gnm_random`, que recibe como parámetros directamente el número de nodos y el número de aristas, lo que resulta cómodo para simular este tipo de redes.

## Algoritmo de flujo máximo

El algoritmo de flujo máximo elegido fue el Edmond Karp, pues fue el que mejores resultados mostró en la tarea anterior. Este algoritmo recibe como parámetros el grafo, el nodo fuente y el nodo sumidero.

## Algoritmo de ordenamiento

El algoritmo de ordenamiento empleado es el propuesto por Fruchterman y Reingold [1]. Este muestra buenos resultados en la distribución de nodos, tamaño de las aristas uniforme y simetría de manera general, enfocándose en la velocidad y simplicidad.

## Generación de grafos

Para realizar el experimento son generados cinco grafos aleatoriamente, con la capacidad de flujo en sus aristas, distribuida normalmente, como puede observarse en la figura 1. El ancho de las aristas es proporcional a su capacidad.

## Características estructurales de los nodos

A cada uno de los grafos se les calculan las siguientes características estructurales de sus nodos:

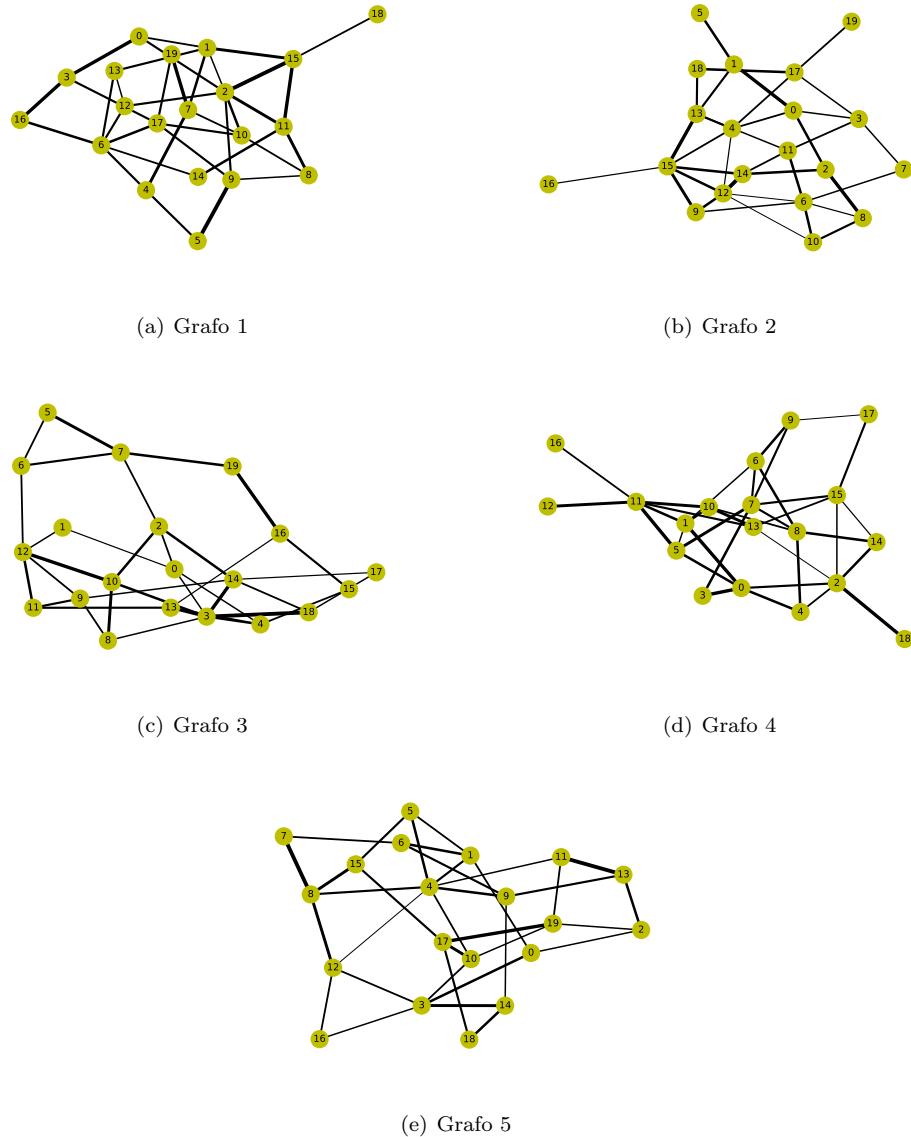


Figura 1: Grafos generados

- Distribución de grado (figura 2)
- Coeficiente de agrupamiento (figura 3)
- Centralidad de cercanía (figura 4)
- Centralidad de carga (figura 5)
- Excentricidad (figura 6)
- Rango de página (figura 7)

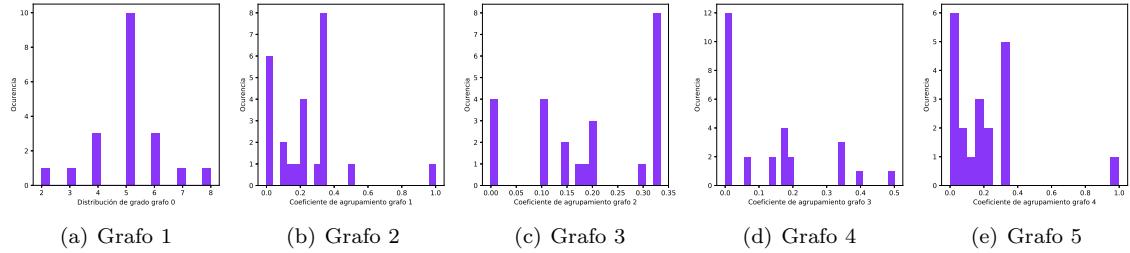


Figura 2: Histogramas de la distribución de grado para cada grafo

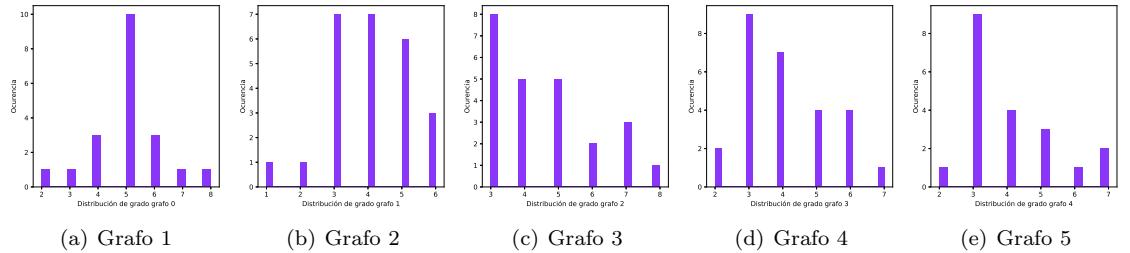


Figura 3: Histogramas del coeficiente de agrupamiento para cada grafo

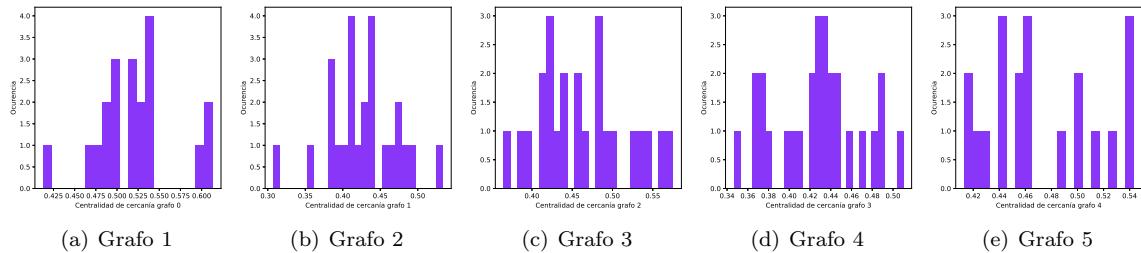


Figura 4: Histogramas de la centralidad de cercanía para cada grafo

## Flujo máximo

Para cada combinación de fuente-sumidero en cada grafo se calcula el flujo máximo. En las figuras 8, 9, 10, 11 y 12, se visualizan los flujos para la mejor y peor pareja de fuente-sumidero para los grafos 1, 2, 3, 4 y 5, respectivamente. Los nodos rojos representan las mejores fuente-sumidero, como los

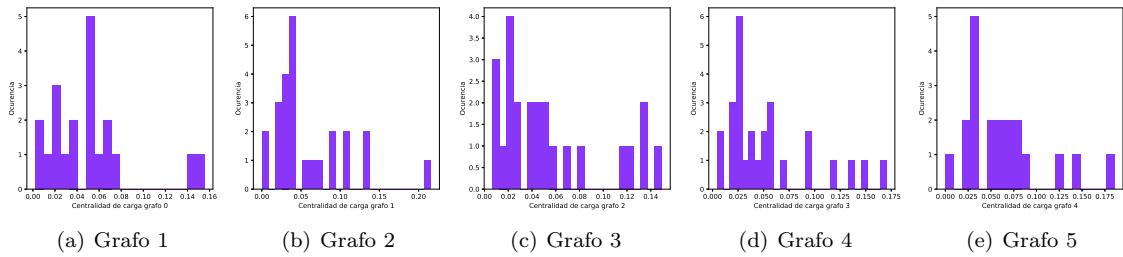


Figura 5: Histogramas de la centralidad de carga para cada grafo

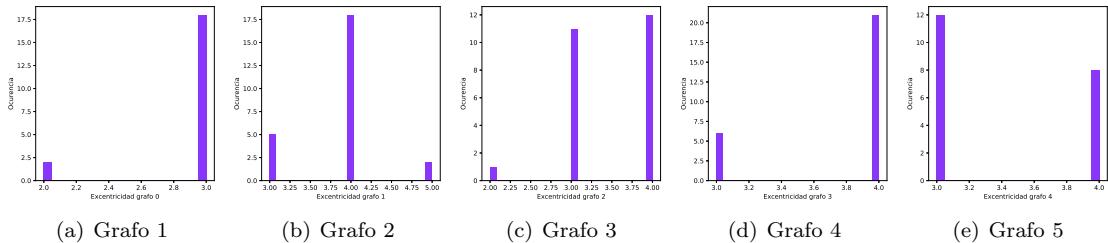


Figura 6: Histogramas de la excentricidad para cada grafo

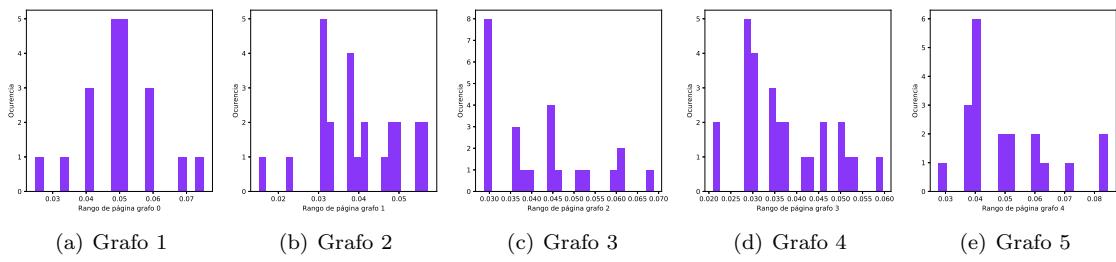


Figura 7: Histogramas de la excentricidad para cada grafo

grafos son no dirigidos, estos son intercambiables. Los nodos azules son buenas fuente-sumidero y los nodos grises son las peores fuentes-sumideros, en cada grafo.

En rosado se muestra cuantos flujo pasa por cada arista una vez aplicado al algoritmo de flujo máximo, aumentando la intensidad del color proporcionalmente a la cantidad de flujo. En las aristas negras no pasa flujo.

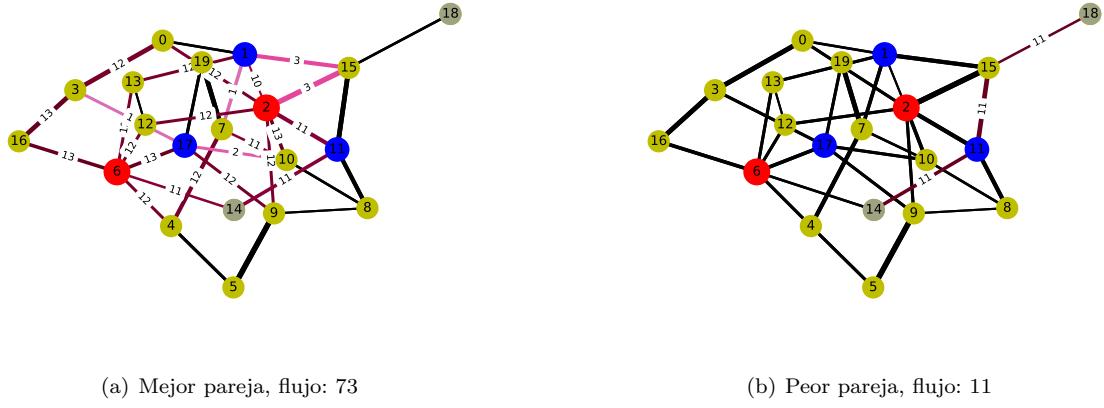


Figura 8: Visualización de la mejor y peor pareja de nodos a usar como fuente o sumidero en el grafo 1

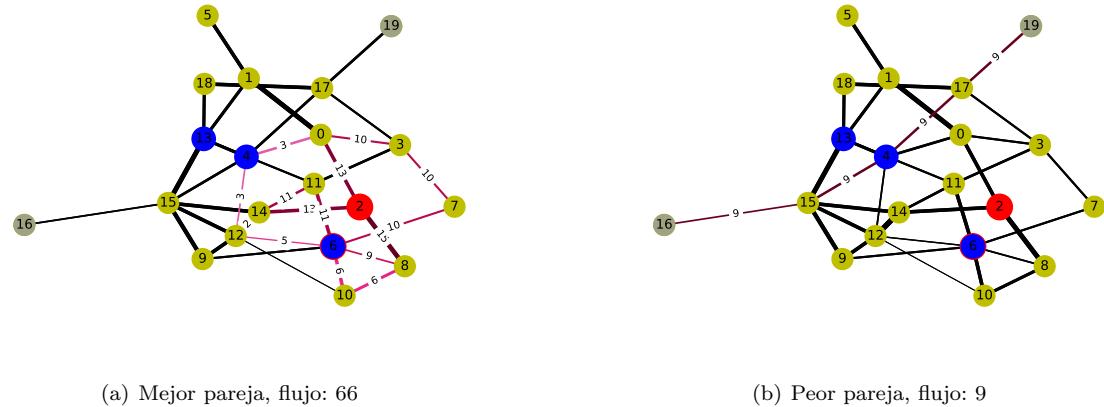


Figura 9: Visualización de la mejor y peor pareja de nodos a usar como fuente o sumidero en el grafo 2

## Influencia de las características estructurales

En el Cuadro 1 se muestran los resultados del análisis de varianza (ANOVA en lo adelante, por sus siglas en inglés) de las características estructurales del grafo para ver su influencia en el tiempo de ejecución del algoritmo de flujo máximo.

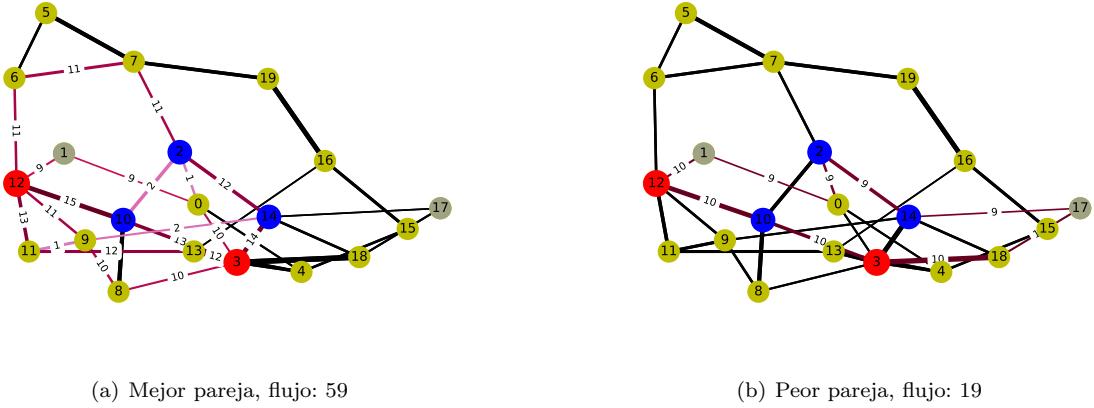


Figura 10: Visualización de la mejor y peor pareja de nodos a usar como fuente o sumidero en el grafo 3

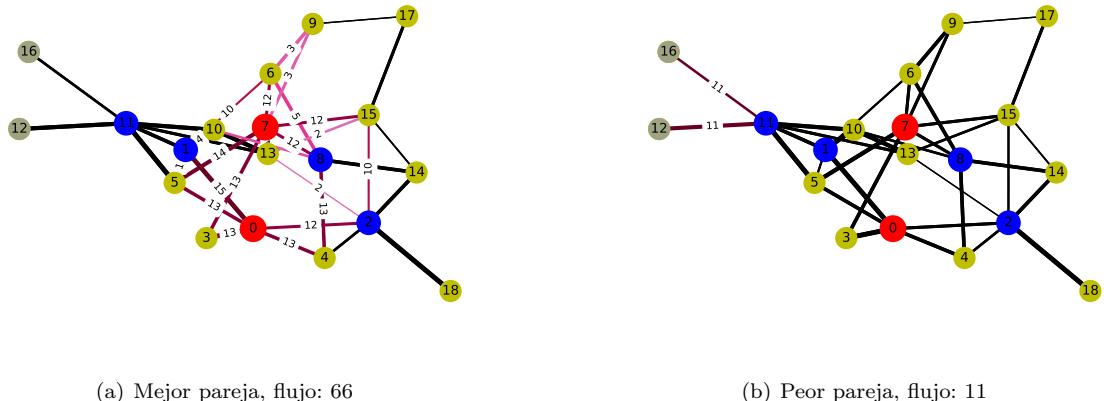
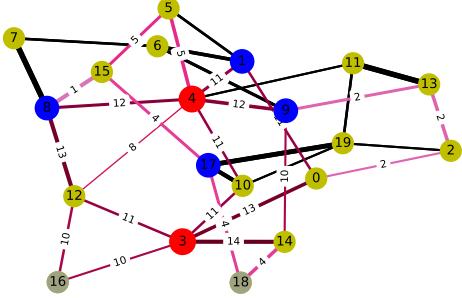
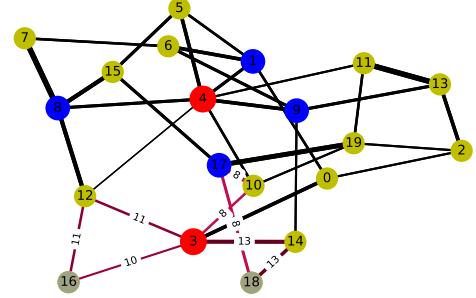


Figura 11: Visualización de la mejor y peor pareja de nodos a usar como fuente o sumidero en el grafo 4



(a) Mejor pareja, flujo: 59



(b) Peor pareja, flujo: 21

Figura 12: Visualización de la mejor y peor pareja de nodos a usar como fuente o sumidero en el grafo 5.

Se aprecia que las características estructurales de los grafos no influyen en el tiempo de ejecución del algoritmo de flujo máximo para los grafos generados. Se observa, además, que existe una relación entre las siguientes características:

- Distribución de grado y coeficiente de agrupamiento
- Distribución de grado y centralidad de cercanía
- Distribución de grado y centralidad de carga
- Rango de página y centralidad de cercanía
- Rango de página y centralidad de carga

Cuadro 1: Resultado del ANOVA para los valores de tiempo

Factor	suma_cuad	F	Prueba
Grado	$8,12 * 10^{-8}$	0.96	0.33
CoefA	$6,78 * 10^{-8}$	0.8	0.37
CentCe	$5,12 * 10^{-9}$	0.06	0.8
CentCa	$2,16 * 10^{-7}$	2.5	0.11
Excent	$2,32 * 10^{-8}$	0.27	0.6
RangoP	$2,11 * 10^{-7}$	2.48	0.11
Grado:CoefA	$4,45 * 10^{-7}$	5.23	0.02
Grado:CentCe	$1,00 * 10^{-6}$	11.8	0.0006
Grado:CentCa	$1,55 * 10^{-6}$	18.28	$2,00 * 10^{-5}$
Grado:Excent	$1,84 * 10^{-7}$	2.16	0.14
Grado:RangoP	$2,30 * 10^{-7}$	2.7	0.1
CentCe:CentCa	$2,11 * 10^{-7}$	2.48	0.12
CentCe:Excent	$1,53 * 10^{-7}$	1.8	0.18
CentCe:RangoP	$7,70 * 10^{-7}$	9.06	0.002
CentCa:Excent	$1,30 * 10^{-7}$	1.53	0.22
CentCa:RangoP	$1,89 * 10^{-6}$	22.25	$2,50 * 10^{-6}$
Excent:RangoP	$1,52 * 10^{-7}$	1.79	0.18
Residual	0.00015678	1844	

## Influencia de las características estructurales del grafo en el flujo máximo

En el Cuadro 1 se muestra el ANOVA de las características estructurales del grafo para analizar su influencia en el flujo máximo.

Se aprecia inciden en el flujo máximo las siguientes características estructurales:

- Coeficiente de agrupamiento
- Centralidad de carga
- Rango de página

Se observa, además, que existe una relación entre las siguientes características:

- Distribución de grado y coeficiente de agrupamiento
- Distribución de grado y rango de página
- Coeficiente de agrupamiento y centralidad de cercanía

- Coeficiente de agrupamiento y rango de página

Cuadro 2: Resultado del ANOVA para los valores de flujo máximo

Factor	sum_cuad	F	Prueba
Grado	38.72	0.5	0.5
CoefA	1363.23	16.16	$6,06 * 10^{-5}$
CentCe	143.62	1.7	0.19
CentCa	976.24	11.57	0
Excent	155.42	1.84	0.17
PageR	508.23	6.02	0.01
Grado:CoefA	650.23	7.7	0
Grado:CentCe	93	1.1	0.29
Grado:CentCa	112.23	1.33	0.25
Grado:Excent	162.65	1.93	0.16
Grado:PageR	660.24	7.82	0.005
CoefA:CentCe	335.67	3.97	0.04
CoefA:CentCa	49.95	0.59	0.44
CoefA:Excent	16.91	0.2	0.65
CoefA:PageR	553.46	6.56	0.01
CentCe:CentCa	7	0.08	0.77
CentCe:Excent	23.2	0.27	0.6
CentCe:PageR	85.71	1.024	0.31
CentCa:Excent	66.43	0.78	0.37
CentCa:PageR	106.07	1.25	0.26
Excent:PageR	104.74	1.24	0.26
Residual	155237.08	1840	

## Código fuente utilizado para realizar el experimento

### Generación de grafos

```

1 def GenerateGraph(nodes ,edges ,address):
2     S=nx.dense_gnm_random_graph(nodes ,edges)
3     scale = 2
4     rang = 10
5     e=S.edges(nbunch=None, data=True, default=None)
6     X = truncnorm(a=rang/scale , b=+rang/scale , scale=scale).rvs(size=edges)
7     X = X.round().astype(int)+rang+2
8     G=nx.Graph()
9     count=0;
10    for i in e:

```

```

11     G.add_edge(i[0], i[1], capacity=X[count])
12     count+=1
13 df = pd.DataFrame()
14 df = nx.to_pandas_adjacency(G, dtype=int, weight='capacity')
15 df.to_csv(address, index=None, header=None)
16
17 def Print(graph, address, pos_address, fig):
18     ds = pd.read_csv(graph, header=None)
19     G = nx.from_pandas_adjacency(ds)
20     pos=nx.fruchterman_reingold_layout(G)
21     X=[]
22     for edge in G.edges():
23         X.append( G.edges[edge]['weight'] )
24     X[:] = [x/10*x/8 for x in X]
25     labels = {}
26     for i in G.nodes:
27         labels[i]=str(i)
28     nx.draw_networkx_nodes(G, pos, node_size=200, node_color='y', node_shape='o')
29     nx.draw_networkx_edges(G, pos, edge_color='black', width=X)
30     nx.draw_networkx_labels(G, pos, labels, font_size=8)
31     plt.axis('off')
32     plt.savefig(fig, dpi=500)
33     df = pd.DataFrame(pos)
34     df.to_csv(address, index=None, header=None)
35     return(G)

```

Grafos.py

## Cálculo de atributos

```

1 dic["Grado"]=[G.degree(i) for i in Nodes]
2 dic["CoefA"]=[nx.clustering(G,i) for i in Nodes]
3 dic["CentCe"]=[nx.closeness_centrality(G,i) for i in Nodes]
4 dic["CentCa"]=[nx.load_centrality(G,i) for i in Nodes]
5 dic["Excent"]=[nx.eccentricity(G,i) for i in Nodes]
6 PageR=nx.pagerank(G,weight="capacity")
7 dic["PageR"]=[PageR[i] for i in Nodes]
8 df=pd.DataFrame(dic)
9 df.to_csv(name, index=None)
10
11 def Time(G,name):
12     dic={"Fuente":[] , "Sumidero":[] , "Media":[] , "Mediana":[] , "Std":[] , "MaxFlow":[]}
13     Nodes=G.nodes;

```

Programa.py

## Cálculo de tiempo de ejecución y flujo máximo

```

1     for j in Nodes:
2         if i!=j:
3             t=[]
4             for k in range(10):
5                 t.append(Edmond(G,i,j))
6             dic["Fuente"].append(i)
7             dic["Sumidero"].append(j)
8             dic["Media"].append(np.mean(t))
9             dic["Mediana"].append(np.median(t))
10            dic["Std"].append(np.std(t))

```

```

11             dic [ "MaxFlow" ] . append ( nx . maximum_flow_value ( G, i , j , capacity=" weight" ) )
12         )
13     df=pd . DataFrame ( dic )
14     df . to_csv ( name , index=None )
15
16 def Histogramas () :
17     cont=0

```

Programa.py

## ANOVA

```

1 modelo = ols ( 'Flujomaximo ~ Grado+CoefA+CentCe+CentCa+Excent+PageR+Grado*CoefA+Grado
   *CentCe+Grado*CentCa+Grado*Excent+Grado*PageR+CentCe*CentCa+CentCe*Excent+CentCe
   *PageR+CentCa*Excent+CentCa*PageR+Excent*PageR' , data=df ) . fit ()
2 print ( modelo . summary () )
3 modelo_csv = open (" Anova_Mult . csv " , 'w' )
4 aov_table = sm . stats . anova_lm ( modelo , typ=2 )
5 df1=pd . DataFrame ( aov_table )

```

procesamiento.py

## Referencias

- [1] Thomas MJ Fruchterman and Edward M Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.