Universidad Autónoma de Nuevo León
Facultad de Ingeniería Mecánica y Eléctrica
Posgrado de Ingeniería en Sistemas

Optimización de flujo en redes
Portafolio de evidencias

Dra Elisa Schaeffer

Liliana Saus
1565171

Semestre enero-junio 2019

7.5

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

Liliana Carolina Saus Olvera 5171

12 de febrero de 2019

Objetivo: Se identifica aplicaciones prácticas para los siguientes tipos de grafos y se representa un ejemplo inspirado en datos reales para cada caso, con por lo menos cinco vértices y no más de doce vértices por caso

1. Grafo simple no dirigido acíclico

1.1. Ejemplo

Las líneas del metro en Nuevo León son una representación de este tipo grafo, en donde las estaciones representan cada uno de los nodos y la ~~distancia~~ entre estas estaciones representan las aristas, una característica de las líneas del metro es que son acíclicas y se mueven en dos direcciones.

1.2. Código

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

G=nx.Graph()
G.add_nodes_from(['1','2','3', '4','5'])
G.add_edges_from([('1','2'),('1','3',)])
G.add_edges_from([('2','4')])
G.add_edges_from([('3','5')])


nx.draw(G, with_labels=True)
plt.show()
```

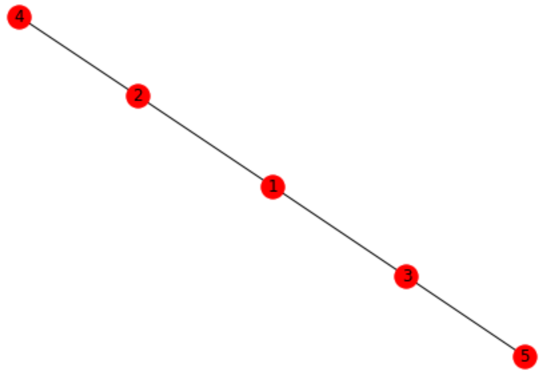


Figura 1: Gráfo simple no dirigido acíclico

2. Gráfo Simple no dirigido cíclico

2.1. Ejemplo

En múltiples lugares de servicio se tienen rutas, las cuales empiezan y terminan en un mismo lugar,esto puede representar un grafo simple no dirigido acíclico, en donde los nodos son los lugares que se debe visitar y las aristas, la distancia de dirigirse a cada lugar.

2.2. Código

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

G=nx.Graph()
G.add_nodes_from(['1','2','3', '4','5'])
G.add_edges_from([('1','2',),('1','5',)])
G.add_edges_from([('2','3',)])
G.add_edges_from([('3','4',)])
G.add_edges_from([('4','5',)])

nx.draw(G, with_labels=True)

plt.show()
```

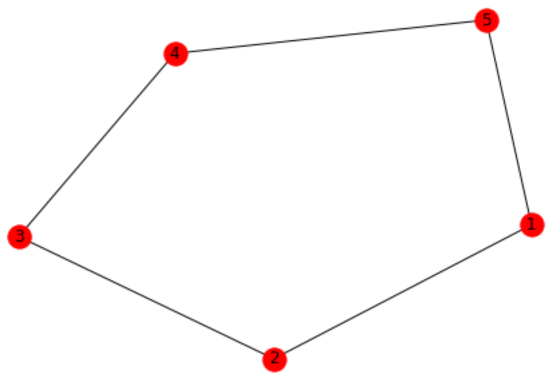


Figura 2: Gráfo simple no dirigido cíclico

3. Simple no dirigido reflexivo

3.1. Código

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



G=nx.Graph()
Vertices={1:(0,0),2:(1,-2),3:(2,-1),4:(3,-2),5:(3,-3)}
Aristas=[(1,2),(1,3),(2,3),(3,4),(4,5),(5,5)]

nx.draw_networkx_nodes(G, Vertices,nodelist = [1,2,3,4], node_color
      = 'b')
nx.draw_networkx_nodes(G, Vertices,nodelist = [5], node_color = 'r'
      )
nx.draw_networkx_edges(G,Vertices,width=1, edgelist=Aristas,alpha
      =1)

plt.axis('off')
```

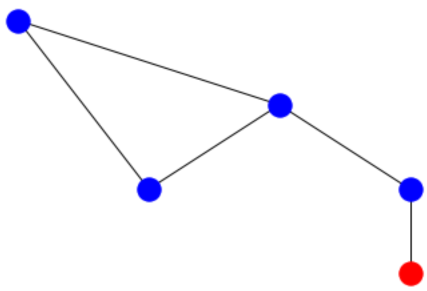


Figura 3: Gráfo simple no dirigido reflexivo

4. Grafo simple dirigido acíclico

~~4.1. Ejemplo~~

Las lineas del metro en Nuevo León representan este tipo de grafo, donde se toma en cuenta solo una dirección del metro, por ejemplo la ruta de ir de la estación Anahuac a Anaya, donde las aristas es la dirección entre cada estación y los nodos son las estaciones.

4.2. Código

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

G=nx.DiGraph()
G.add_nodes_from(["1","2","3","4","5"])

G.add_edges_from([("1","2"),("2","3"),("3","4"),("4","5")])
nx.draw(G, with_labels=True)
plt.savefig("Graph2.eps", format="EPS")
plt.show()
```

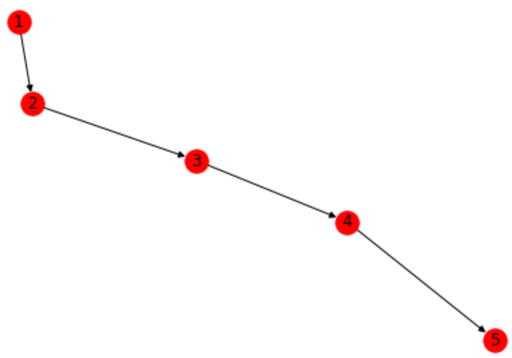


Figura 4: Gráfo simple dirigido acíclico

5. Grafo simple dirigido cíclico

5.1. Ejemplo

La ruta del camión 213 es un ejemplo, ya que su recurrido es un ciclo y las paradas consecutivas representan los nodos y la distancia entre estas paradas las aristas.

5.2. Código

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

```

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

nx.draw(G, with_labels=True)

plt.show()
```

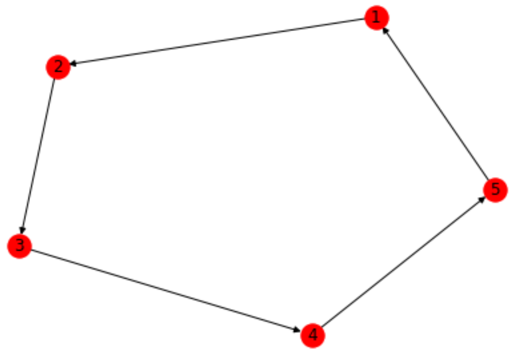


Figura 5: Gráfo simple dirigido cíclico

6. Grafo simple dirigido reflexivo

6.1. Ejemplo

La órbita de la tierra representa este tipo de grafo, donde el movimiento que hace en girar en si misma, da lugar a reflexividad.

6.2. Código

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



G=nx.Graph()
Vertices={1:(0,0),2:(1,-2),3:(2,-1),4:(3,-2),5:(3,-3)}
Aristas=[(1,2),(3,1),(2,3),(4,3),(5,4),(5,5)]

nx.draw_networkx_nodes(G, Vertices, nodelist = [1,2,3,4], node_color
    = 'b')
nx.draw_networkx_nodes(G, Vertices, nodelist = [5], node_color = 'r'
    )
nx.draw_networkx_edges(G,Vertices,width=1, edgelist=Aristas,alpha
    =1)

plt.axis('off')
```

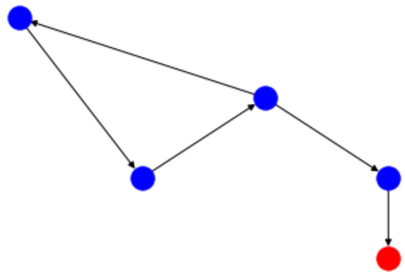


Figura 6: Gráfo simple reflexivo

7. Multigrafo no dirigido acíclico

7.1. Ejemplo

La linea 1 y la linea 2 del metro, su recorrido representa un multigrafo no dirigido aciclico.

7.2. Código

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

G=nx.Graph()
G.add_nodes_from(['1','2','3', '4','5'])
G.add_edges_from([('1','2'),('1','3',)])
G.add_edges_from([('2','4')])
G.add_edges_from([('3','5')])
G.add_edges_from([('5','3')])


nx.draw_random(G, with_labels=True)
plt.show()
```

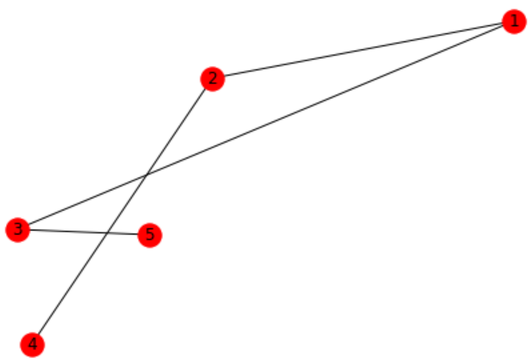


Figura 7: Multigrafo no dirigido acíclico

8. Multigrafo no dirigido cíclico

8.1. Ejemplo

En ocasiones existen distintas alternativas para llegar a un mismo lugar, como lo es la vialidad en Nuevo León, el cual podemos consider como un multigrafo no dirigido aciclico.

8.2. Código

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

G=nx.Graph()
Vertices={1:(0,0),2:(-1,-1),3:(1,-1),4:(2,-1),5:(3,0),6:(4,-1)}
Aristas=[(1,2),(3,2),(5,3),(4,3),(5,4),(5,6),(6,4)]

nx.draw_networkx_nodes(G, Vertices, nodelist = [1,2,3,4,5,6],
    node_color = 'b')
nx.draw_networkx_edges(G,Vertices,width=1, edgelist=Aristas,alpha
    =1)

plt.axis('off')
```

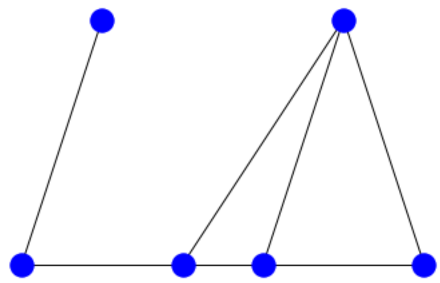


Figura 8: Multigrafo no dirigido cíclico

9. Multigrafo no dirigido reflexivo

9.1. Código

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

G=nx.Graph()
Vertices={1:(0,0),2:(-1,-1),3:(1,-1),4:(2,-1),5:(3,0),6:(4,-1)}
Aristas=[(1,2),(3,2),(5,3),(4,3),(5,4),(5,6),(6,4)]

nx.draw_networkx_nodes(G, Vertices, nodelist = [1,2,3,4,5],
    node_color = 'b')
nx.draw_networkx_nodes(G, Vertices, nodelist = [6], node_color = 'r'
    )
```

```
nx.draw_networkx_edges(G,Vertices,width=1, edgelist=Aristas,alpha
    =1)

plt.axis('off')
```

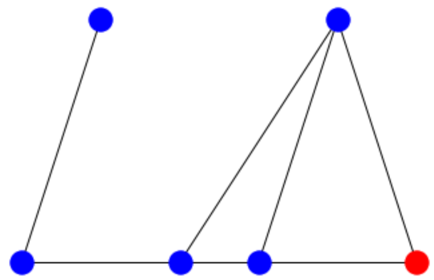


Figura 9: Multigrafo no dirigido reflexivo

10. Multigrafo dirigido acíclico

10.1. Ejemplo

La red de tuberías de pluvial, donde los nodos son puntos de presión para impulsar el agua hacia los destinos y las son las tuberías.

10.2. Código

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

G=nx.DiGraph()
Vertices={1:(0,0),2:(-1,-1),3:(1,-1),4:(2,-1),5:(2,0),6:(4,-1)
    ,7:(5,-1)}
Aristas=[(3,1),(2,1),(3,2),(3,4),(2,4),(4,5),(7,5),(6,5)]

nx.draw_networkx_nodes(G, Vertices,nodelist = [1,2,3,4,5,6,7],
    node_color = 'b')
nx.draw_networkx_edges(G,Vertices,width=1, edgelist=Aristas,alpha
    =1)

plt.axis('off')
```

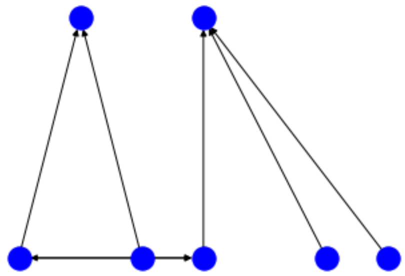


Figura 10: Multigrafo dirigido acíclico

11. Multigrafo dirigido cíclico

11.1. Código

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

G=nx.DiGraph()
Vertices={1:(0,0),2:(-1,-1),3:(1,-1),4:(2,-1),5:(3,0),6:(4,-1)}
Aristas=[(1,2),(3,2),(3,5),(4,3),(5,4),(5,6),(6,4)]

nx.draw_networkx_nodes(G, Vertices,nodelist = [1,2,3,4,5,6],
    node_color = 'b')
nx.draw_networkx_edges(G,Vertices,width=1, edgelist=Aristas,alpha
    =1)

plt.axis('off')
```

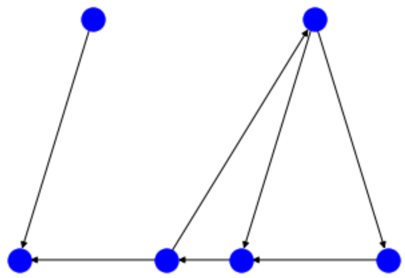


Figura 11: Multigrafo dirigido cíclico

12. Multigrafo dirigido reflexivo

12.1. Código

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

```

G=nx.DiGraph()
Vertices={1:(0,0),2:(-1,-1),3:(1,-1),4:(2,-1),5:(3,0),6:(4,-1)}
Aristas=[(1,2),(3,2),(5,3),(4,3),(5,4),(5,6),(6,4)]

nx.draw_networkx_nodes(G, Vertices,nodelist = [1,2,3,4,5],
    node_color = 'b')
nx.draw_networkx_nodes(G, Vertices,nodelist = [6], node_color = 'r'
    )
nx.draw_networkx_edges(G,Vertices,width=1, edgelist=Aristas,alpha
    =1)

plt.axis('off')
```

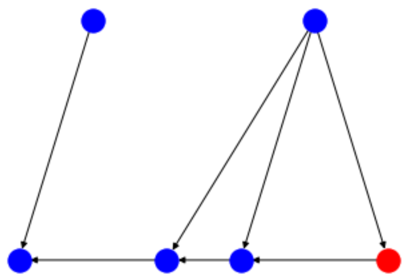


Figura 12: Multigrafo dirigido reflexivo

Referencias

[1] SCHAEFFER E. *Optimización de flujo en redes, 2019.*
`https://elisa.dyndns-web.com/teaching/opt/flow/`

Bernardes A. Com

network

Tarea 1

Al realizar el reporte de la actividad indicada, se obtuvo una retroalimentación, de la cual se hicieron las siguientes correcciones: se agregaron ejemplos para todos los grafos, se realizaron correcciones sobre signos de puntuación, se eliminaron los espacios entre cada sección enumerada, se corrigió ortografía de la descripción de todas las imágenes y se agregaron referencias bibliográficas utilizadas.

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

5171

1 de junio de 2019

Objetivo: Se identifican aplicaciones prácticas para los siguientes tipos de grafos y se representa un ejemplo inspirado en datos reales para cada caso, con por lo menos cinco vértices y no más de doce vértices por caso.

1. Grafo simple no dirigido acíclico

Ejemplo:

Las líneas del metro en Nuevo León son una representación de este tipo grafo, en donde las estaciones representan cada uno de los nodos y la distancia entre estas estaciones representan las aristas, una característica de las líneas del metro es que son acíclicas y se mueven en dos direcciones.

Código:

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

G=nx.Graph()
G.add_nodes_from(['1','2','3', '4','5'])
G.add_edges_from([('1','2'),('1','3',)])
G.add_edges_from([('2','4')])
G.add_edges_from([('3','5')])


nx.draw(G, with_labels=True)
plt.show()
```

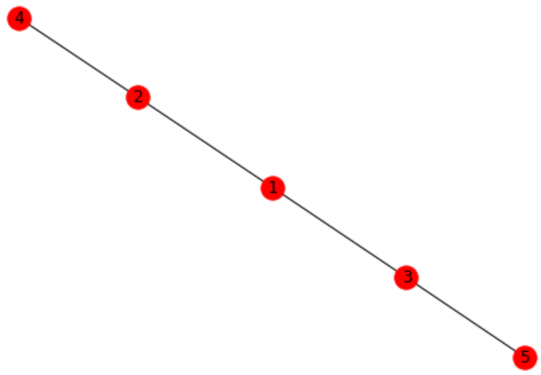


Figura 1: Grafo simple no dirigido acíclico

2. Grafo Simple no dirigido cíclico

Ejemplo:

En múltiples lugares de servicio se tienen rutas, las cuales empiezan y terminan en un mismo lugar, esto puede representar un grafo simple no dirigido acíclico, en donde los nodos son los lugares que se debe visitar y las aristas, la distancia de dirigirse a cada lugar.

Código:

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

G=nx.Graph()
G.add_nodes_from(['1','2','3'], bipartite=0)
G.add_nodes_from(['4','5'], bipartite=1)
G.add_edges_from([('1','5',)])
G.add_edges_from([('2','4',)])


nx.draw(G, with_labels=True)

plt.show()
```

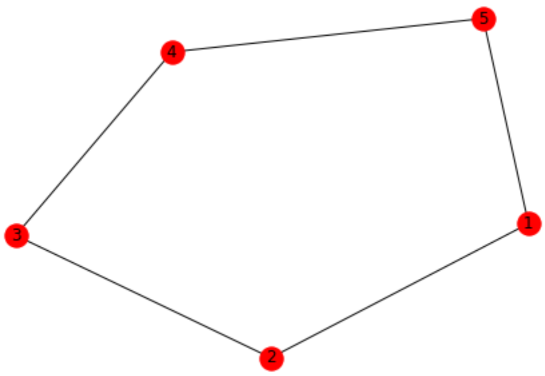


Figura 2: Grafo simple no dirigido cíclico

3. Simple no dirigido reflexivo

Ejemplo:

El grafo representa la relación social que existe entre las personas, como es el caso que se pueden unir tres nodos, que es la relación de amistad entre ellos, mientras que uno de ellos tiene amistad con alguien externo a los restantes, como se puede ver en la siguiente figura.

Código:

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



G=nx.Graph()
Vertices={1:(0,0),2:(1,-2),3:(2,-1),4:(3,-2),5:(3,-3)}
Aristas=[(1,2),(1,3),(2,3),(3,4),(4,5),(5,5)]

nx.draw_networkx_nodes(G, Vertices,nodelist = [1,2,3,4], node_color = 'b')
nx.draw_networkx_nodes(G, Vertices,nodelist = [5], node_color = 'r')
nx.draw_networkx_edges(G,Vertices,width=1, edgelist=Aristas,alpha=1)

plt.axis('off')
```

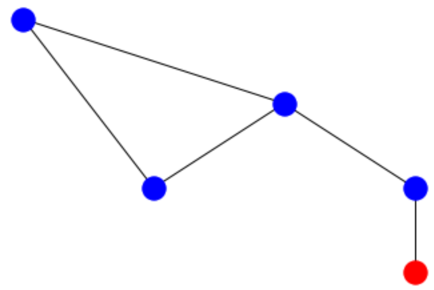


Figura 3: Grafo simple no dirigido reflexivo

4. Grafo simple dirigido acíclico

Ejemplo:

Las líneas del metro en Nuevo León representan este tipo de grafo, donde se toma en cuenta solo una dirección del metro, por ejemplo la ruta de ir de la estación Anáhuac a Anaya, donde las aristas es la dirección entre cada estación y los nodos son las estaciones.

Código:

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

G=nx.DiGraph()
G.add_nodes_from(["1","2","3","4","5"])

G.add_edges_from([("1","2"),("2","3"),("3","4"),("4","5")])
nx.draw(G,with_labels=True)
plt.savefig("Graph2.eps", format="EPS")
plt.show()
```

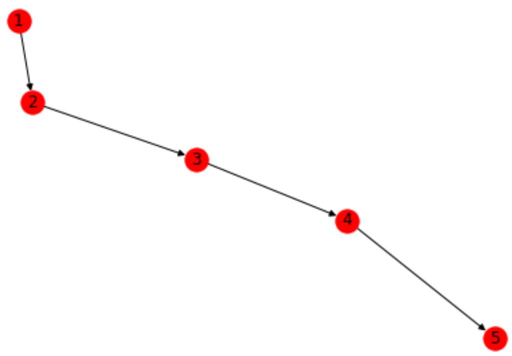


Figura 4: Grafo simple dirigido acíclico

5. Grafo simple dirigido cíclico

Ejemplo:

La ruta del camión 213 es un ejemplo, ya que su recurrido es un ciclo y las paradas consecutivas representan los nodos y la distancia entre estas paradas las aristas.

Código

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

G=nx.DiGraph()
G.add_nodes_from(['1','2','3', '4','5'])
G.add_edges_from([('1','2',),('5','1',)])
G.add_edges_from([('2','3',)])
G.add_edges_from([('3','4',)])
G.add_edges_from([('4','5',)])

nx.draw(G, with_labels=True)

plt.show()
```

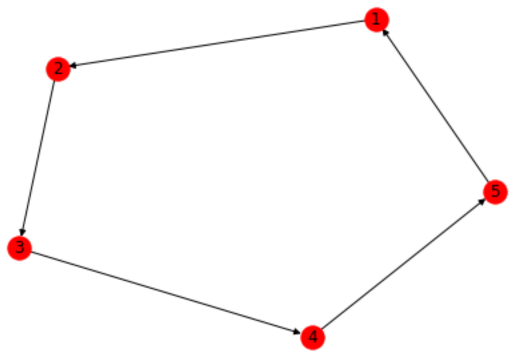


Figura 5: Grafo simple dirigido cíclico

6. Grafo simple dirigido reflexivo

Ejemplo:

La órbita de la tierra representa este tipo de grafo, donde el movimiento que hace en girar en si misma, da lugar a reflexividad.

Código:

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

```

```
G=nx.Graph()
Vertices={1:(0,0),2:(1,-2),3:(2,-1),4:(3,-2),5:(3,-3)}
Aristas=[(1,2),(3,1),(2,3),(4,3),(5,4),(5,5)]

nx.draw_networkx_nodes(G, Vertices,nodelist = [1,2,3,4], node_color
    = 'b')
nx.draw_networkx_nodes(G, Vertices,nodelist = [5], node_color = 'r'
    )
nx.draw_networkx_edges(G,Vertices,width=1, edgelist=Aristas,alpha
    =1)

plt.axis('off')
```

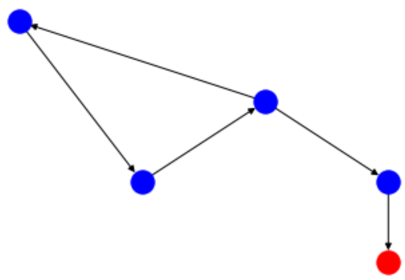


Figura 6: Grafo simple reflexivo

7. Multigrafo no dirigido acíclico

Ejemplo:

La línea 1 y la línea 2 del metro, su recorrido representa un, multigrafo no dirigido aciclico.

Código:

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

G=nx.Graph()
G.add_nodes_from(['1','2','3', '4','5'])
G.add_edges_from([('1','2'),('1','3',)])
G.add_edges_from([('2','4')])
G.add_edges_from([('3','5')])
G.add_edges_from([('5','3')])


nx.draw_random(G,with_labels=True)
plt.show()
```

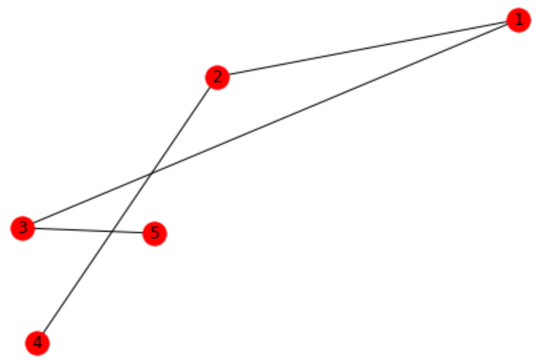


Figura 7: Multigrafo no dirigido acíclico

8. Multigrafo no dirigido cíclico

Ejemplo:

En ocasiones existen distintas alternativas para llegar a un mismo lugar, como lo es la vialidad en Nuevo León, el cual podemos consider como un multigrafo no dirigido aciclico.

Código:

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

G=nx.Graph()
Vertices={1:(0,0),2:(-1,-1),3:(1,-1),4:(2,-1),5:(3,0),6:(4,-1)}
Aristas=[(1,2),(3,2),(5,3),(4,3),(5,4),(5,6),(6,4)]

nx.draw_networkx_nodes(G, Vertices, nodelist = [1,2,3,4,5,6],
    node_color = 'b')
nx.draw_networkx_edges(G,Vertices,width=1, edgelist=Aristas,alpha
    =1)

plt.axis('off')
```

9. Multigrafo no dirigido reflexivo

Ejemplo:

En una escuela se aplica un examen a ciertos alumnos de diferentes escuelas, donde las aristas indican con quién competirá cada escuela, donde uno de las reglas es, si se puede competir con su misma escuela.

Código:

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

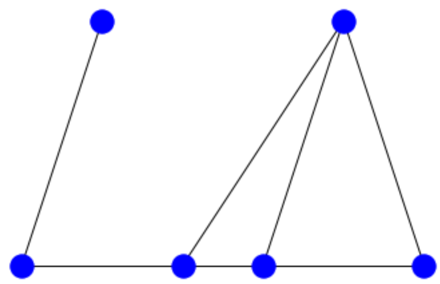


Figura 8: Multigrafo no dirigido cíclico

```

G=nx.Graph()
Vertices={1:(0,0),2:(-1,-1),3:(1,-1),4:(2,-1),5:(3,0),6:(4,-1)}
Aristas=[(1,2),(3,2),(5,3),(4,3),(5,4),(5,6),(6,4)]

nx.draw_networkx_nodes(G, Vertices,nodelist = [1,2,3,4,5],
    node_color = 'b')
nx.draw_networkx_nodes(G, Vertices,nodelist = [6], node_color = 'r'
    )
nx.draw_networkx_edges(G,Vertices,width=1, edgelist=Aristas,alpha
    =1)

plt.axis('off')
```

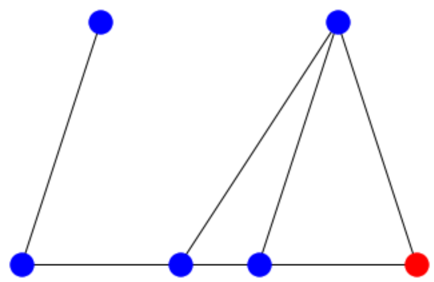


Figura 9: Multigrafo no dirigido reflexivo

10. Multigrafo dirigido acíclico

10.1. Ejemplo:

La red de tuberías de pluvial, donde los nodos son puntos de presión para impulsar el agua hacia los destinos y las son las tuberías.

10.2. Código:

```
import networkx as nx
```

```
import matplotlib.pyplot as plt

G=nx.DiGraph()
Vertices={1:(0,0),2:(-1,-1),3:(1,-1),4:(2,-1),5:(2,0),6:(4,-1)
    ,7:(5,-1)}
Aristas=[(3,1),(2,1),(3,2),(3,4),(2,4),(4,5),(7,5),(6,5)]

nx.draw_networkx_nodes(G, Vertices, nodelist = [1,2,3,4,5,6,7],
    node_color = 'b')
nx.draw_networkx_edges(G,Vertices,width=1, edgelist=Aristas,alpha
    =1)

plt.axis('off')
```

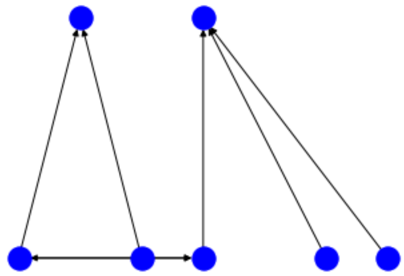


Figura 10: Multigrafo dirigido acíclico

11. Multigrafo dirigido cíclico

Ejemplo:

Este grafo representa el viaje que una persona puede realizar, donde los nodos representan los lugares que visita y las aristas el trayecto que puedes tomar, tomando en cuenta que las rutas presentan una dirección señalada.

11.1. Código:

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

G=nx.DiGraph()
Vertices={1:(0,0),2:(-1,-1),3:(1,-1),4:(2,-1),5:(3,0),6:(4,-1)}
Aristas=[(1,2),(3,2),(3,5),(4,3),(5,4),(5,6),(6,4)]

nx.draw_networkx_nodes(G, Vertices, nodelist = [1,2,3,4,5,6],
    node_color = 'b')
nx.draw_networkx_edges(G,Vertices,width=1, edgelist=Aristas,alpha
    =1)

plt.axis('off')
```

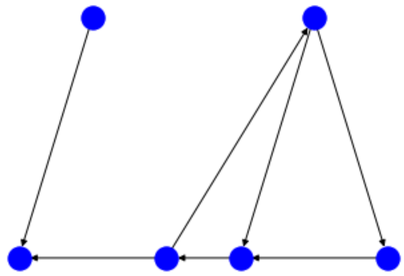


Figura 11: Multigrafo dirigido cíclico

12. Multigrafo dirigido reflexivo

Ejemplo:

Este grafo representa el viaje que una persona puede realizar, donde los nodos representan los paises que visita y las aristas el trayecto que puedes tomar, es importante resaltar que puede visitar las ciudades dentro de su país.

12.1. Código:

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

G=nx.DiGraph()
Vertices={1:(0,0),2:(-1,-1),3:(1,-1),4:(2,-1),5:(3,0),6:(4,-1)}
Aristas=[(1,2),(3,2),(5,3),(4,3),(5,4),(5,6),(6,4)]

nx.draw_networkx_nodes(G, Vertices,nodelist = [1,2,3,4,5],
    node_color = 'b')
nx.draw_networkx_nodes(G, Vertices,nodelist = [6], node_color = 'r'
    )
nx.draw_networkx_edges(G,Vertices,width=1, edgelist=Aristas,alpha
    =1)

plt.axis('off')
```

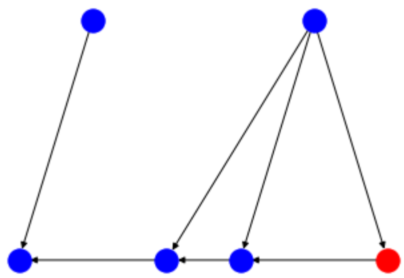


Figura 12: Multigrafo dirigido reflexivo

Referencias

[1] Elisa S. *Optimización de flujo en redes, 2019.*
https://elisa.dyndns-web.com/teaching/opt/flow/

[2] Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart *Exploring network structure, dynamics, and function using NetworkX, 2008.*
https://networkx.github.io/documentation/stable/index.html

5.5

Tarea 2

5171

26 de febrero de 2019

1. Algoritmo de acomodo Bipartito

Descripción Se tienen dos conjuntos de nodos y un conjunto de aristas, que conectan los nodos unicamente los nodos de los conjuntos opuestos.

Código

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

G=nx.Graph()
G.add_nodes_from(['1','2','3'], bipartite=0)
G.add_nodes_from(['4','5'], bipartite=1)
G.add_edges_from([('1','5',)])
G.add_edges_from([('2','4',)])
G.add_edges_from([('2','5',)])
G.add_edges_from([('3','5',)])
G.add_edges_from([('3','4',)])
G.add_edges_from([('1','4',)])
nx.draw(G,pos=nx.bipartite_layout(G,['1','2','3']), with_labels=
    True)
plt.savefig("1.eps")
plt.show()
```

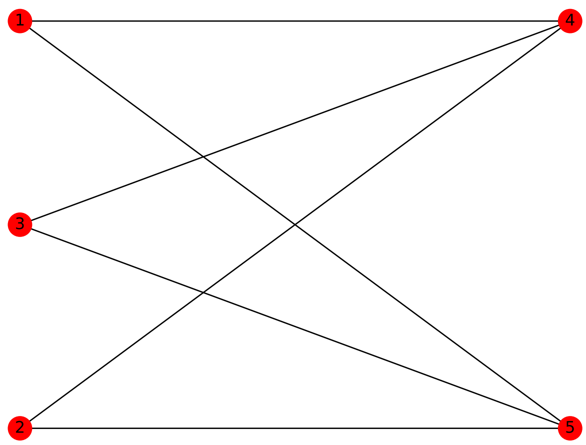


Figura 1: Grafo simple no dirigido acíclico

2. Algoritmo de acomodo circular

Descripción Este tipo de acomodo consiste en colocar los nodos de forma circular.

2.1. Código

```
import networkx as nx

import matplotlib.pyplot as plt

G=nx.Graph()
G.add_nodes_from(['1','2','3', '4','5'])
G.add_edges_from([('1','2'),('2','3',)])
G.add_edges_from([('3','4')])
G.add_edges_from([('4','5')])
G.add_edges_from([('5','1')])


nx.draw_circular(G,with_labels=True)
plt.show()
```

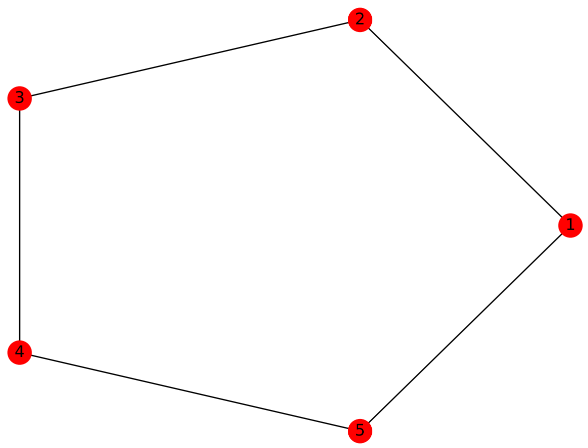


Figura 2: Grafo simple no dirigido cíclico

3. Algoritmo de acomodo kamada kawai

Descripción Este tipo de acomodo funciona mejor con grafos ponderados, consiste en agrupar los nodos con aristas con mayor fuerza.

3.1. Código

```
from typing import Dict, Tuple

import networkx as nx

import matplotlib.pyplot as plt

```

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

colores=[]
for node in G:
    if(node==5):
        colores.append('red')
    else:
        colores.append('yellow')

acomodo=nx.kamada_kawai_layout(G)
nx.draw(G, pos=acomodo, node_color=colores, with_labels=True)

plt.show()
```

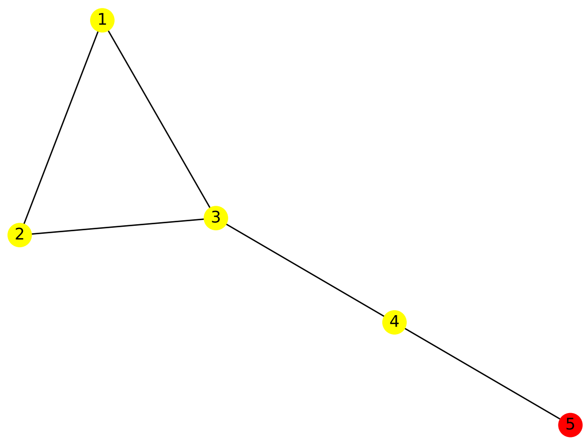


Figura 3: Grafo simple no dirigido reflexivo

4. Algoritmo de acomodo aleatorio.

Descripción Consiste en ubicar los nodos de forma aleatoria uniforme en un cuadrado.

4.1. Código

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

G=nx.DiGraph()

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

nx.draw_random(G, with_labels=True)

plt.savefig("Graph2.eps", format="EPS")

plt.show()
```

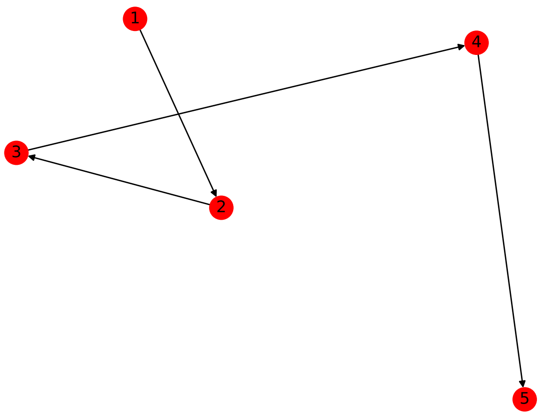


Figura 4: Grafo simple dirigido acíclico

5. Algoritmo Shell

Descripción Consiste en acomodar los nodos en circulos concentricos.

5.1. Código

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

G: DiGraph=nx.DiGraph()
G.add_nodes_from(['1','2','3', '4','5'])
G.add_edges_from([('1','2',),('5','1',)])
G.add_edges_from([('2','3',)])
G.add_edges_from([('3','4',)])
G.add_edges_from([('4','5',)])

nx.draw_shell(G, with_labels=True)

plt.show()
```

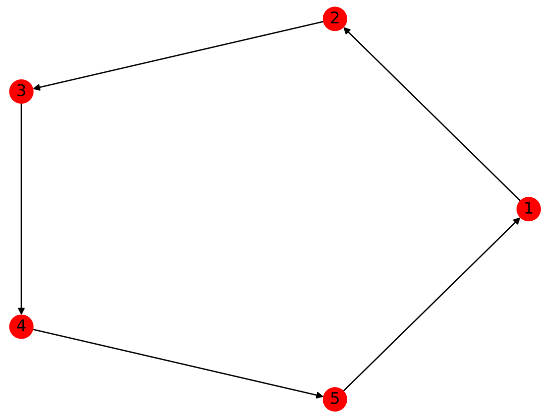


Figura 5: Gráfo simple dirigido cíclico

6. Algoritmo de acomodo de resorte

Descripción Este acomodo posiciona los nodos utilizando el algoritmo de Fruchterman-Reingold. Los nodos están representados por anillos de acero y las aristas son resortes entre ellas. La fuerza de atracción es análoga a la fuerza de resorte y la fuerza de repulsión es análoga a la fuerza eléctrica.

6.1. Código

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

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

colores=[]
for node in G:
    if(node==5):
        colores.append('red')
    else:
        colores.append('yellow')


nx.draw_spring(G, node_color=colores, with_labels=True)

plt.show()
```

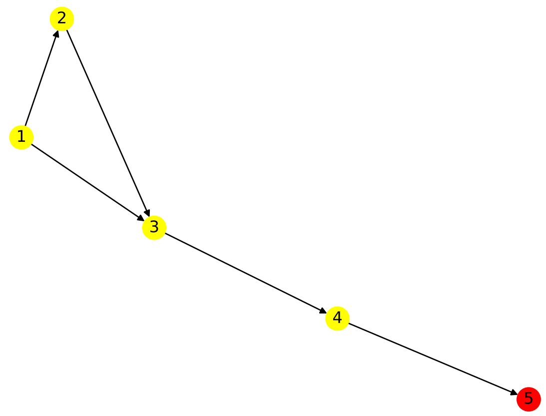


Figura 6: Gráfo simple dirigido reflexivo

7. Algortimo de acomodo de espectro

Descripción Consiste en calcular los dos valores propios más grandes (o más pequeños) y los vectores propios correspondientes de la matriz laplaciana del grafo y luego usarlos para colocar realmente los nodos. subsectionCódigo

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

G=nx.MultiGraph()
G.add_nodes_from(['1','2','3', '4','5'])
G.add_edges_from([('1','2'),('1','3',)])
G.add_edges_from([('2','4')])
G.add_edges_from([('4','2')])
G.add_edges_from([('3','5')])
G.add_edges_from([('5','3')])

nx.draw_spectral(G,with_labels=True)

plt.show()
```

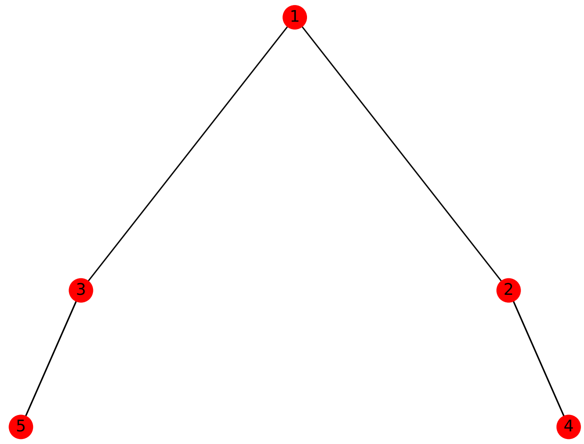


Figura 7: Multigrafo no dirigido acíclico

BibTeX

Referencias

[1] Schaeffer E. *Optimización de flujo en redes, 2019.*
`https://elisa.dyndns-web.com/teaching/opt/flow/`

[2] Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart *Exploring network structure, dynamics, and function using NetworkX,2008*
`Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), (Pasadena, CA USA)`

editor = { — and — — —

address = { — — }

Tarea 2

Al realizar el reporte de la actividad indicada, se obtuvo retroalimentación, de la cual se hicieron las siguientes correcciones: se agregaron los grafos faltantes, los de la tarea 1, utilizando un diferente algoritmo de acomodo, se hicieron correcciones de ortografía y correcciones en la bibliografía.

Tarea 2

5171

2 de junio de 2019

Algoritmo de acomodo bipartito

Descripción Se tienen dos conjuntos de nodos y un conjunto de aristas, que conectan los nodos, unicamente los nodos de los conjuntos opuestos.

Grafo simple no dirigido acíclico

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

G=nx.Graph()
G.add_nodes_from(['1','2','3'], bipartite=0)
G.add_nodes_from(['4','5'], bipartite=1)
G.add_edges_from([('1','5',)])
G.add_edges_from([('2','4',)])
G.add_edges_from([('2','5',)])
G.add_edges_from([('3','5',)])
G.add_edges_from([('3','4',)])
G.add_edges_from([('1','4',)])
nx.draw(G,pos=nx.bipartite_layout(G,['1','2','3']), with_labels=
    True)
plt.savefig("1.eps")
plt.show()
```

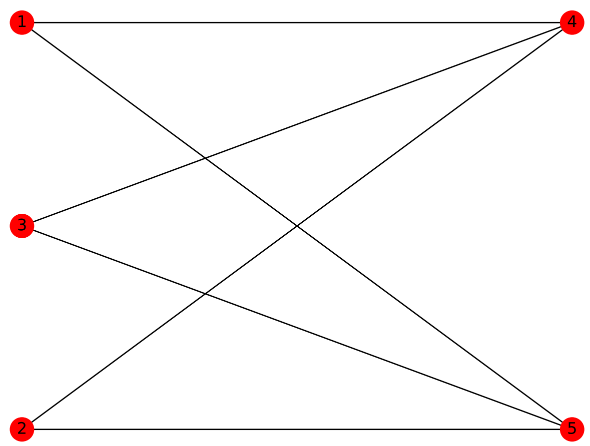


Figura 1: Grafo simple no dirigido acíclico

Multigrafo no dirigido reflexivo

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

G= nx.MultiGraph()
Vertices={1:(0,0),2:(-1,-1),3:(1,-1),4:(2,-1),5:(3,0),6:(4,-1)}
Aristas=[(1,2),(3,2),(5,3),(4,3),(5,4),(5,6),(6,4)]
G.add_edges_from(Aristas)
G.add_nodes_from(Vertices)

nx.draw(G, pos=nx.bipartite_layout(G, [1,6,3]), with_labels=True)
plt.show()
```

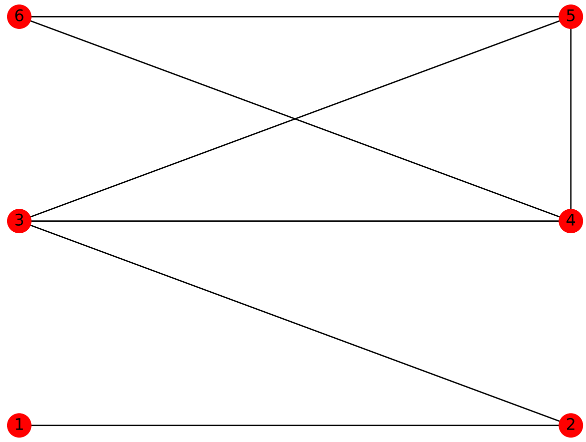


Figura 2: Multigrafo no dirigido reflexivo

Algoritmo de acomodo circular

Descripción Este tipo de acomodo consiste en colocar los nodos de forma circular.

Grafo simple no dirigido cíclico

```
import networkx as nx

import matplotlib.pyplot as plt

G=nx.Graph()
G.add_nodes_from(['1','2','3', '4','5'])
G.add_edges_from([('1','2'),('2','3',)])
G.add_edges_from([('3','4')])
G.add_edges_from([('4','5')])
G.add_edges_from([('5','1')])


nx.draw_circular(G,with_labels=True)
plt.show()
```

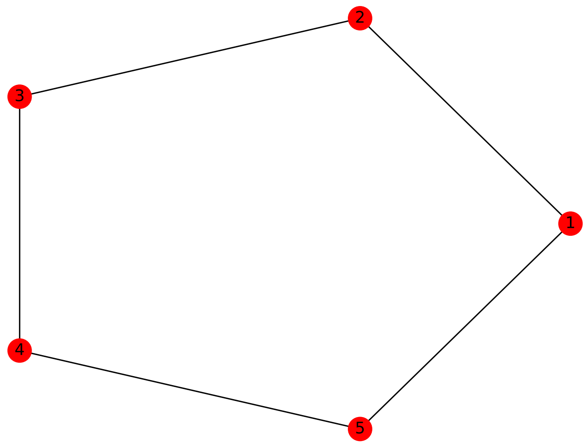


Figura 3: Grafo simple no dirigido cíclico

Multigrafo no dirigido cíclico

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

G=nx.Graph()
Vertices={1:(0,0),2:(-1,-1),3:(1,-1),4:(2,-1),5:(3,0),6:(4,-1)}
Aristas=[(1,2),(3,2),(5,3),(4,3),(5,4),(5,6),(6,4)]

G.add_nodes_from(Vertices)
G.add_edges_from(Aristas)

nx.draw_networkx_nodes(G, Vertices,nodelist = [1,2,3,4,5,6],
    node_color = 'b')
nx.draw_networkx_edges(G,Vertices,width=1, edgelist=Aristas,alpha
    =1)
nx.draw(G, pos=nx.circular_layout(G), with_labels=True)
plt.axis('off')
```

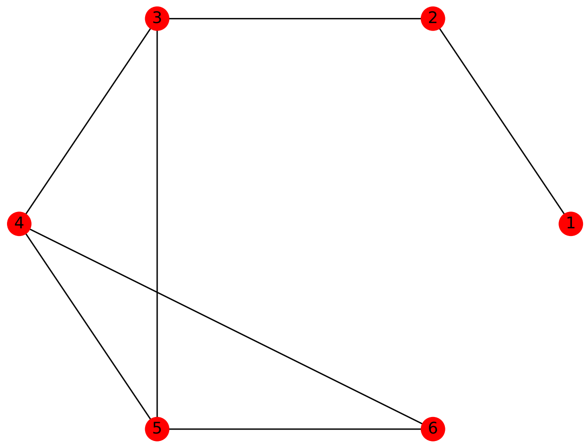


Figura 4: Multigrafo no dirigido cíclico

Algoritmo de acomodo Kamada Kawai

Descripción Este tipo de acomodo funciona mejor con grafos ponderados, consiste en agrupar los nodos con aristas con mayor fuerza.

```
from typing import Dict, Tuple

import networkx as nx

import matplotlib.pyplot as plt

G=nx.Graph()
G.add_nodes_from([1,5])
G.add_edges_from([(1,2),(1,3),(2,3),(3,4),(4,5),(5,5)])

colores=[]
for node in G:
    if(node==5):
        colores.append('red')
    else:
        colores.append('yellow')

acomodo=nx.kamada_kawai_layout(G)
nx.draw(G, pos=acomodo, node_color=colores, with_labels=True)

plt.show()
```

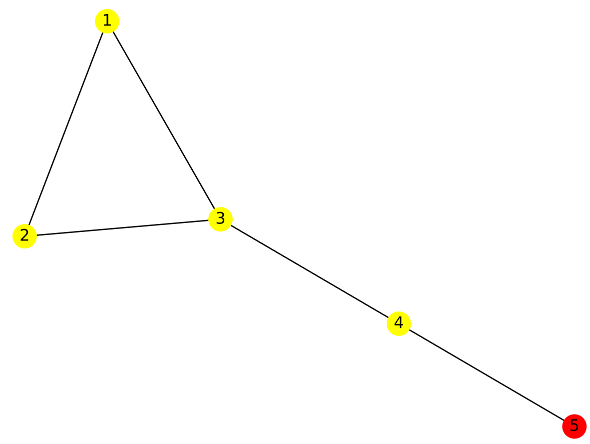


Figura 5: Grafo simple no dirigido reflexivo

Algoritmo de acomodo aleatorio

Descripción Consiste en ubicar los nodos de forma aleatoria uniforme en un cuadrado.

Grafo simple dirigido acíclico

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

G=nx.DiGraph()

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

nx.draw_random(G, with_labels=True)

plt.savefig("Graph2.eps", format="EPS")

plt.show()
```

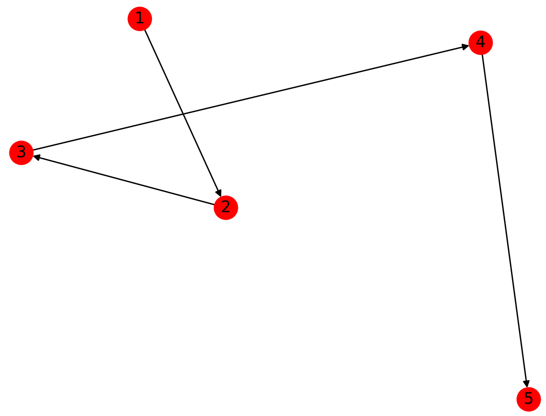


Figura 6: Grafo simple dirigido acíclico

Multigrafo dirigido acíclico

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

G=nx.MultiGraph()
Vertices={1:(0,0),2:(-1,-1),3:(1,-1),4:(2,-1),5:(2,0),6:(4,-1)
    ,7:(5,-1)}
Aristas = [(3,1),(2,1),(3,2),(3,4),(2,4),(4,5),(7,5),(6,5)]

G.add_edges_from(Aristas)
G.add_nodes_from(Vertices)
nx.draw_random(G, with_labels=True)
plt.show()
```

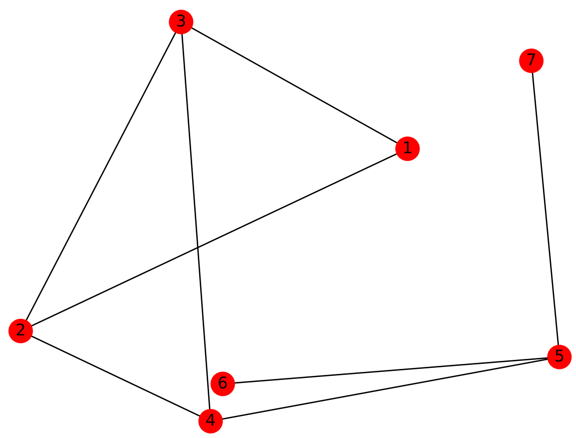


Figura 7: Multigrafo dirigido acíclico

Algoritmo de cascaron

Descripción Consiste en acomodar los nodos en círculos concéntricos.

Grafo simple dirigido cíclico

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

G: DiGraph=nx.DiGraph()
G.add_nodes_from(['1','2','3', '4','5'])
G.add_edges_from([('1','2',),('5','1',)])
G.add_edges_from([('2','3',)])
G.add_edges_from([('3','4',)])
G.add_edges_from([('4','5',)])

nx.draw_shell(G, with_labels=True)

plt.show()
```

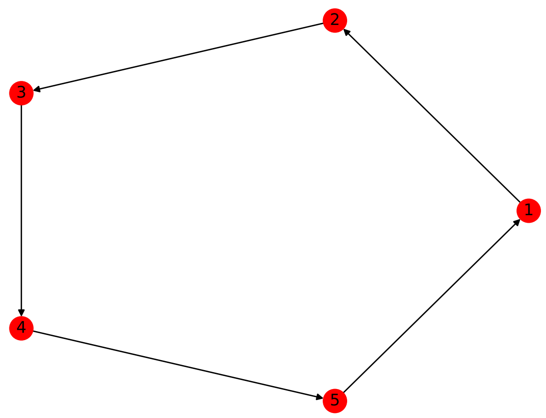


Figura 8: Gráfo simple dirigido cíclico

Multigrafo dirigido cíclico

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

G=nx.DiGraph()
Vertices={1:(0,0),2:(-1,-1),3:(1,-1),4:(2,-1),5:(3,0),6:(4,-1)}
Aristas=[(1,2),(3,2),(3,5),(4,3),(5,4),(5,6),(6,4)]

G.add_nodes_from(Vertices)
G.add_edges_from(Aristas)
nx.draw_shell(G, with_labels=True)

plt.axis('off')
```

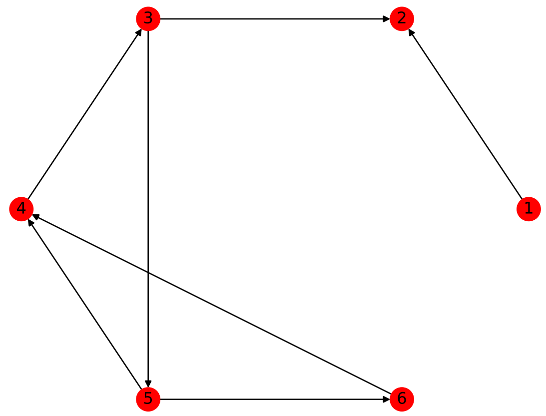


Figura 9: Multigrafo dirigido cíclico

Algoritmo de acomodo de resorte

Descripción Este acomodo posiciona los nodos utilizando el algoritmo de Fruchterman-Reingold. Los nodos están representados por anillos de acero y las aristas son resortes entre ellas. La fuerza de atracción es análoga a la fuerza de resorte y la fuerza de repulsión es análoga a la fuerza eléctrica.

Gráfo simple dirigido reflexivo

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

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

colores=[]
for node in G:
    if(node==5):
        colores.append('red')
    else:
        colores.append('yellow')


nx.draw_spring(G, node_color=colores, with_labels=True)

plt.show()
```

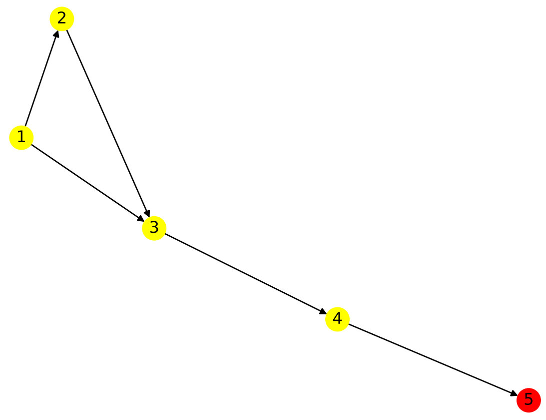


Figura 10: Gráfo simple dirigido reflexivo

Multigrafo dirigido reflexivo

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

G=nx.DiGraph()
Vertices={1:(0,0),2:(-1,-1),3:(1,-1),4:(2,-1),5:(3,0),6:(4,-1)}
Aristas=[(1,2),(3,2),(5,3),(4,3),(5,4),(5,6),(6,4)]

G.add_edges_from(Aristas)
G.add_nodes_from(Vertices)
nx.draw_spectral(G, with_labels=True)
plt.axis('off')
```

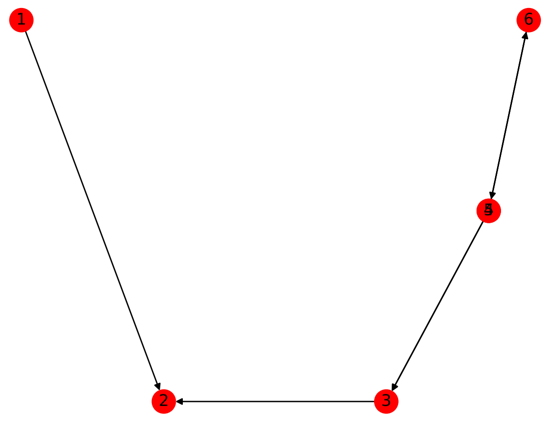


Figura 11: Multigrafo dirigido reflexivo

Algoritmo de acomodo de espectro

Descripción Consiste en calcular los dos valores propios más grandes (o más pequeños) y los vectores propios correspondientes de la matriz laplaciana del grafo y luego usarlos para colocar realmente los nodos.

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

G=nx.MultiGraph()
G.add_nodes_from(['1','2','3', '4','5'])
G.add_edges_from([('1','2'),('1','3',)])
G.add_edges_from([('2','4')])
G.add_edges_from([('4','2')])
G.add_edges_from([('3','5')])
G.add_edges_from([('5','3')])

nx.draw_spectral(G,with_labels=True)

plt.show()
```

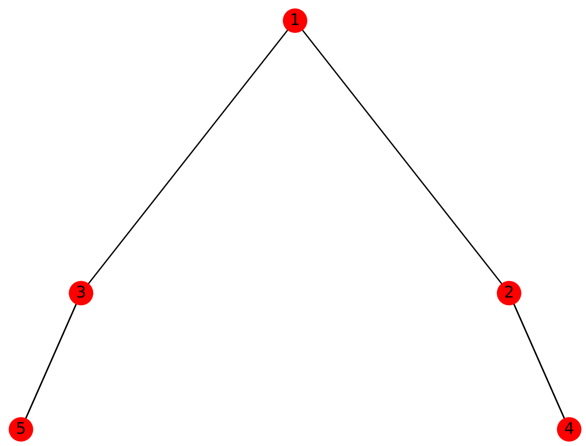


Figura 12: Multigrafo no dirigido acíclico

Referencias

[1] Elisa S. *Optimización de flujo en redes, 2019.*
`https://elisa.dyndns-web.com/teaching/opt/flow/`

[2] Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart *Exploring network structure, dynamics, and function using NetworkX, 2008*
`https://networkx.github.io/`

9

Optimización de flujo en redes

5171
Tarea #3

19 de marzo de 2019

Utilizando las algoritmos para grafos de NetworkX, se implementa un código en Python que ejecuta los siguientes cinco algoritmos.

1. All shortest paths
2. Betweenness centrality
3. Depth first search tree
4. Greedy color
5. Maximal weight matching

1. All shortest paths

Este algoritmo encuentra las longitudes de las rutas más cortas entre todos los pares de vértices.

1.1. Código

```
tiempos = []
for i in range(30):
    start = tm.time()
    for x in range(8000000):
        nx.all_shortest_paths(G, source='1', target='3')
    end = tm.time()
    tiempos.append(end - start)
```

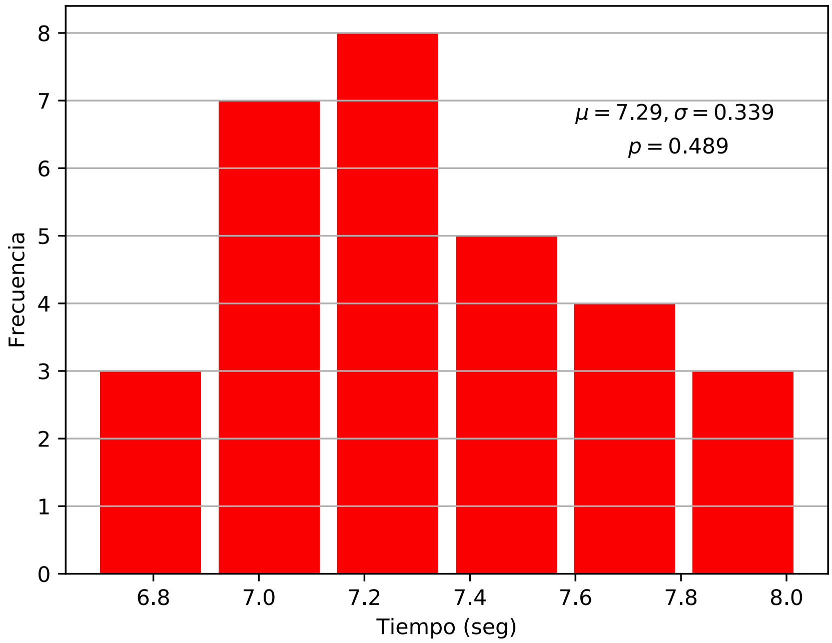


Figura 1: Histrograma.

2. Betweenness centrality

Para cada par de vértices en un gráfico conectado, existe al menos una ruta más corta entre los vértices, de manera tal que el número de bordes por los que pasa el camino se minimiza. La centralidad intermedia entre cada vértice es el número de estos caminos más cortos que pasan a través del vértice.

2.1. Código

```
tiempo2 = []
for i in range(30):
    start = tm.time()
    for x in range(8000000):
        nx.betweenness_centrality(G, normalized=True)
    end = tm.time()
    tiempo2.append(end - start)
```

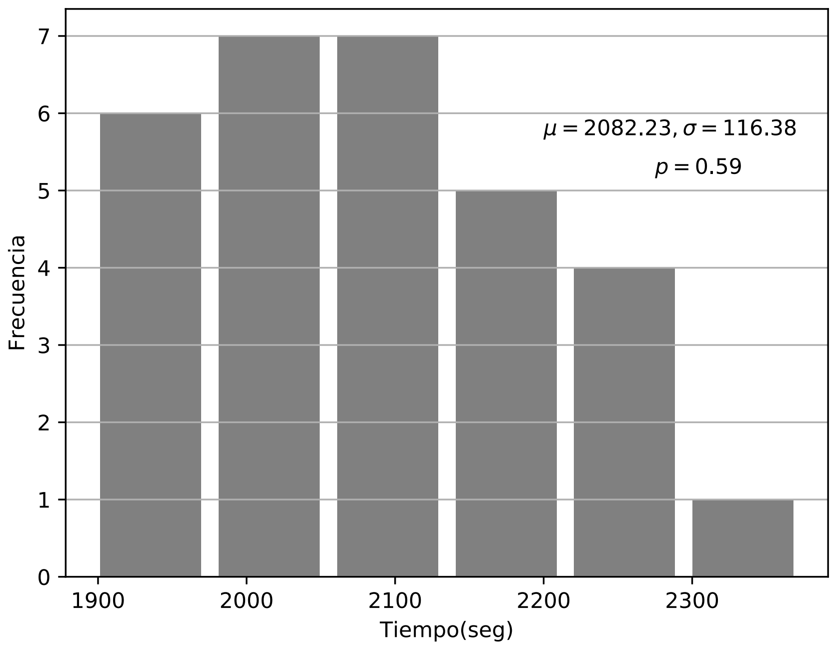


Figura 2: Histrograma.

3. Depth first search tree

Es un árbol orientado construido a partir de una fuente de búsqueda en profundidad.

3.1. Código

```
tiempo3 = []
for i in range(30):
    start = tm.time()
    for x in range(8000000):
        nx.dfs_tree(G, source=0)
    end = tm.time()
    tiempo3.append(end - start)
```

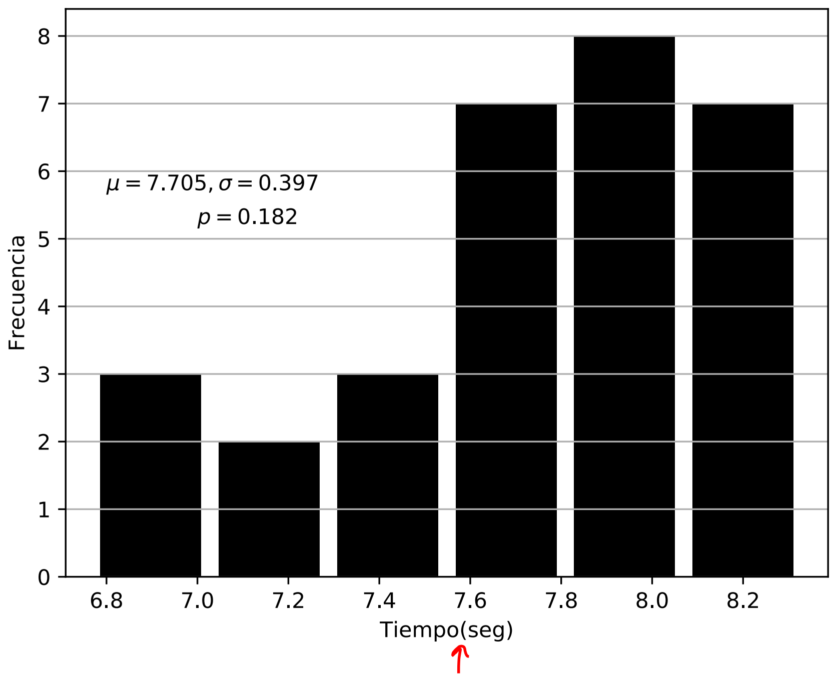


Figura 3: Histrograma.

4. Greedy color

Intenta colorear una gráfica con la menor cantidad de colores posible, donde ningún vecino de un nodo puede tener el mismo color que el nodo mismo. La estrategia dada determina el orden en que se colorean los nodos.

4.1. Código

```
tiempo4 = []
for i in range(30):
    start = tm.time()
    for x in range(8000000):
        nx.greedy_color(G, strategy='largest_first')
    end = tm.time()
    tiempo4.append(end - start)
```

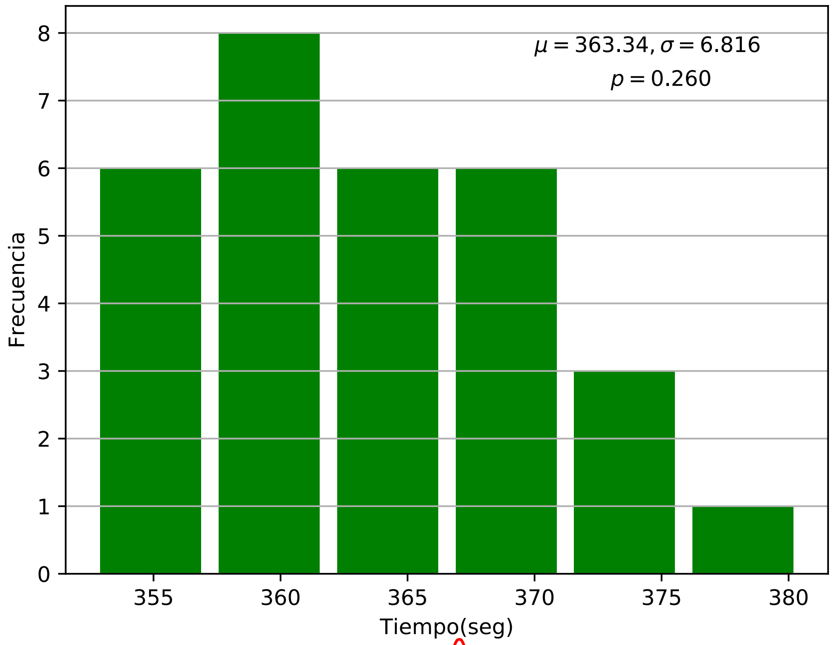


Figura 4: Histrograma.

5. Maximal weight matching

Calcula una coincidencia máxima ponderada de G. Una coincidencia es un subconjunto de bordes en los que no se produce ningún nodo más de una vez.

5.1. Código

```
tiempo5 = []
for i in range(30):
    start = tm.time()
    for x in range(8000000):
        nx.max_weight_matching(G)
    end = tm.time()
    tiempo5.append(end - start)
```

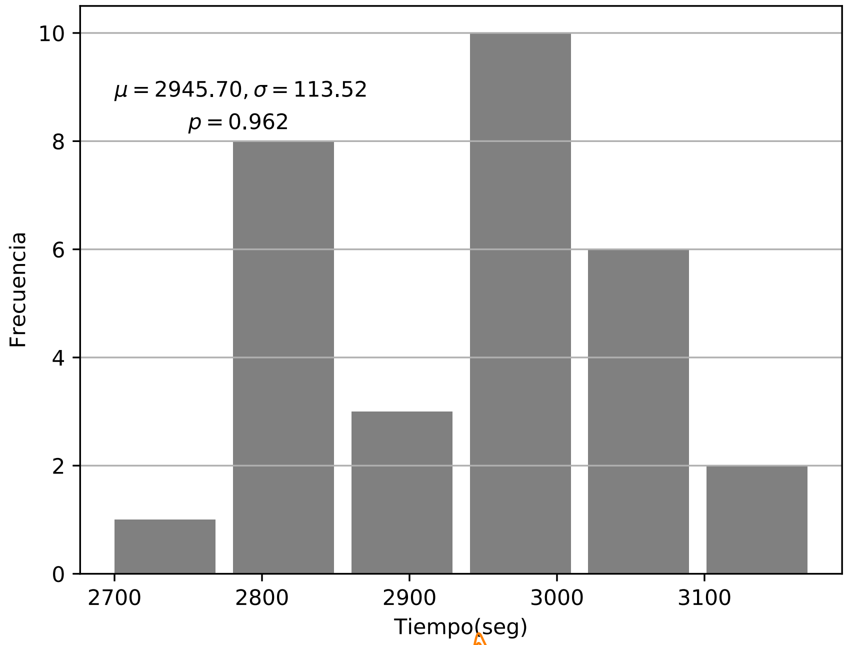


Figura 5: Histrograma.

En cada una de las figuras anteriores se les realizó a los datos la prueba de *Shapiro*, la cual arrojó normalidad para los tiempos de ejecución de la implementación de todos los algoritmos.

En las figuras 6 y 7, se observa como los algoritmos más tardados son el betweenness centrality y maximal weight matching.

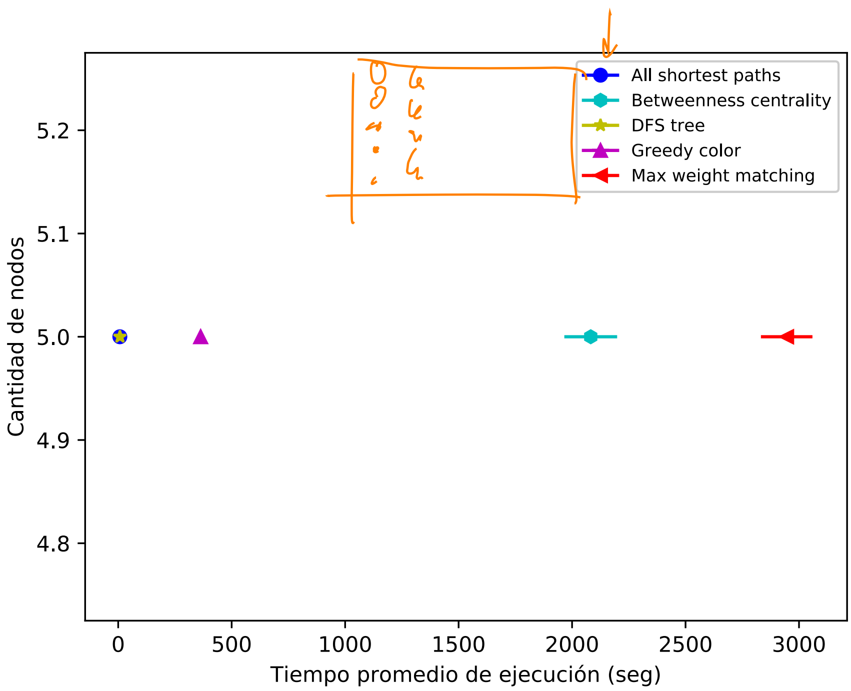


Figura 6: Diagrama de dispersión.

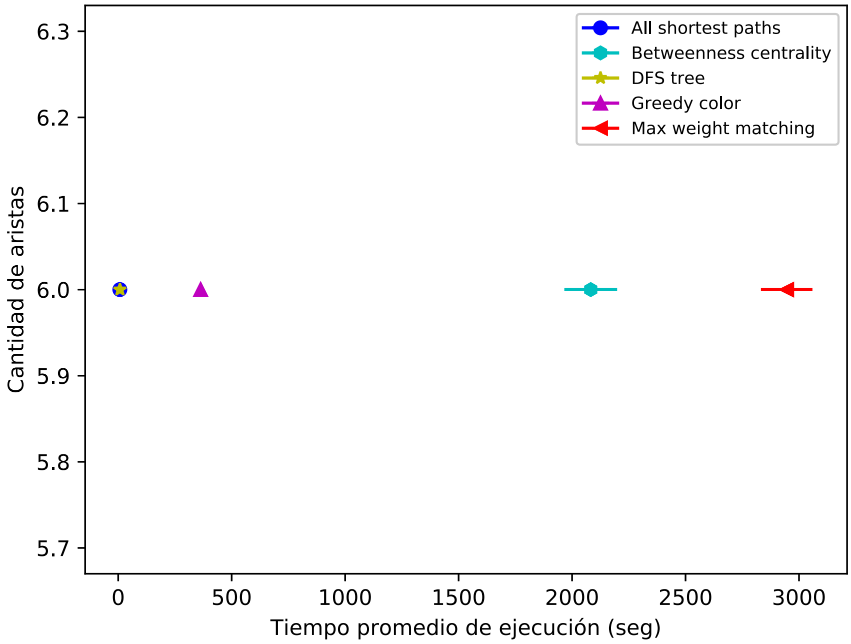


Figura 7: Diagrama de dispersión.

Referencias

[1] Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart *Exploring network structure, dynamics, and function using NetworkX*
`https://networkx.github.io/documentation/stable/`

[2] Schaeffer E. *Optimización de flujo en redes, 2019.*
`https://elisa.dyndns-web.com/teaching/opt/flow/`

[3] Tomihisa Kamada, Satoru Kawai. *An Algorithm for Drawing General Undirected Graphs*
`Information Processing Letters, 1988.`

10

Tarea 4: Complejidad asintótica experimental

5171

2 de abril de 2019

Objetivo: Utilizando los generadores de grafos de NetworkX, se selecciona por lo menos tres métodos de generación de grafos. Con cada generador, se generan cuatro diferentes órdenes en escala logarítmica (16, 32, 64, 128) y 10 grafos distintos de cada orden. Se le asigna pesos no-negativos normalmente distribuidos a las aristas para que se puedan utilizar como instancias del problema de flujo máximo.
Eligiendo por lo menos tres implementaciones de NetworkX de los algoritmos de flujo máximo, se ejecuta los algoritmos seleccionados con cinco diferentes pares de fuente-sumidero. Con métodos estadísticos y visualizaciones científicas se determina:

- el efecto que el generador de grafo usado tiene en el tiempo de ejecución.
- el efecto que el algoritmo usado tiene en el tiempo de ejecución.
- el efecto que el orden del grafo tiene en el tiempo de ejecución.
- el efecto que la densidad del grafo tiene en el tiempo de ejecución.

1. Generadores implementados

A continuación se describe los 3 métodos de generación de grafos.

1.1. Complete

tres

Devuelve el grafo completo con n nodos.

1.2. Wheel

Devuelve el grafo: un solo nodo central conectado a cada nodo del gráfico de ciclo del nodo $(n-1)$. Las etiquetas de nodo son los números enteros de 0 a $n-1$.

1.3. Cycle

Devuelve el grafo de ciclo C_n sobre n nodos. C_n es la ruta n con dos nodos finales conectados.

2. Algoritmos implementados

A continuación se describe los algoritmos de flujo máximo implementados, que fueron elegidos por el menor tiempo de ejecución.

2.1. Edmons

Encuentra un flujo máximo de un solo producto utilizando el algoritmo Edmonds-Karp. Esta función devuelve la red residual resultante después de calcular el flujo máximo.

Este algoritmo tiene un tiempo de ejecución de $O(nm^2)$ para n nodos y m aristas.

2.2. Preflow

Encuentra un flujo máximo de un solo producto utilizando el algoritmo de empuje previo al flujo de la etiqueta más alta. Este algoritmo tiene un tiempo de ejecución de $\mathrm{O}(n^2\sqrt{m})$ para los nodos n y m aristas.

2.3. Dinitz

Encuentra un flujo máximo de un solo producto utilizando el algoritmo de Dinitz.

$\mathcal{O}$ ....

$\mathcal{O}$

3. Resultados

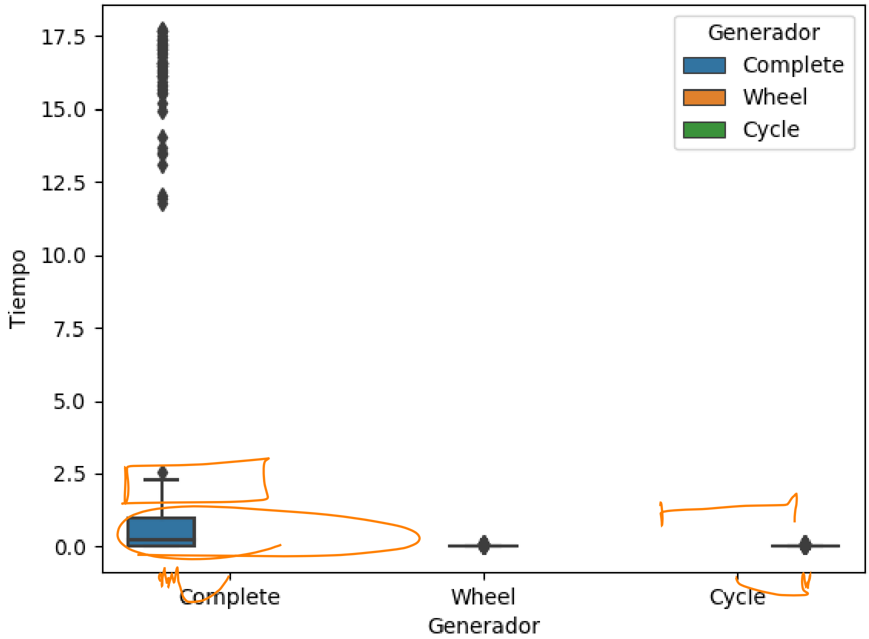


Figura 1: Efecto del generador de grafo en el tiempo de ejecución (segundos).

Se puede observar en la figura 1 que el generador que toma más tiempo en su ejecución es el generador Complete, mientras que en los otros dos sus tiempos son muy parecidos.

\em

\textit

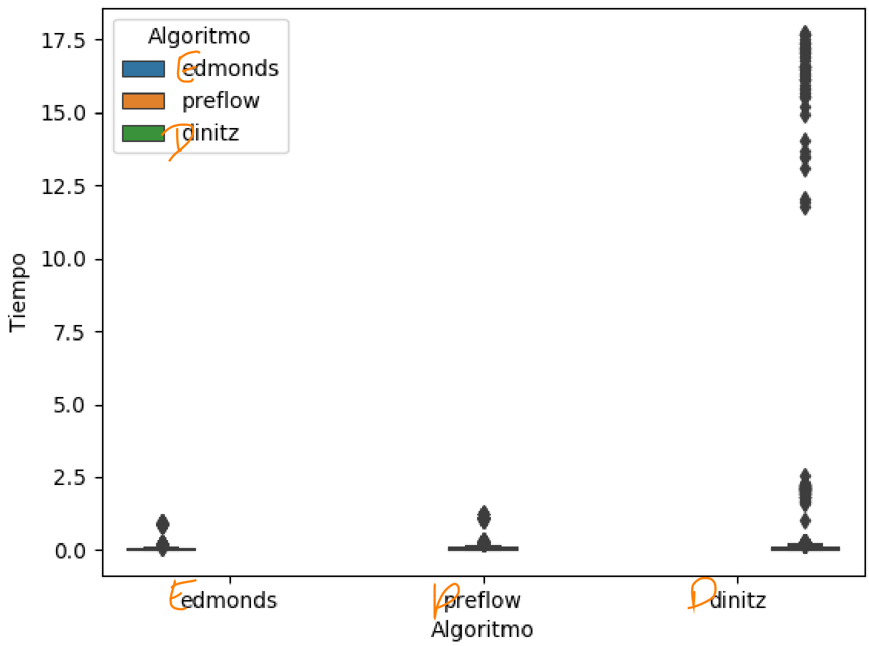


Figura 2: Efecto del algoritmo en el tiempo de ejecución (segundos).

En la figura 2 los tiempos de ejecución de los algoritmos son muy parecidos, por lo que se tendría que realizar un análisis estadístico para concluir del posible efecto.

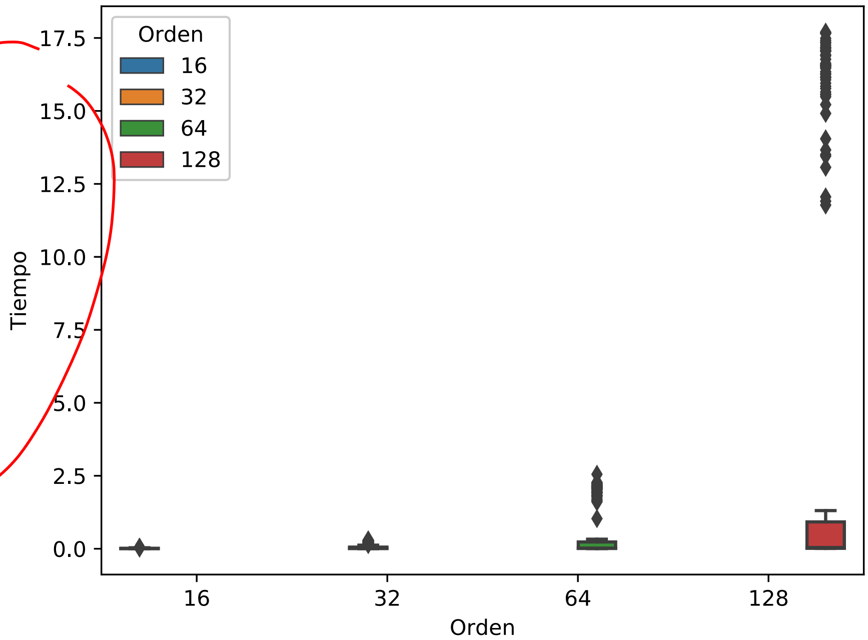


Figura 3: Efecto del orden del grafo en el tiempo de ejecución (segundos).

En la figura 3 se puede observar como ligeramente los tiempos van aumentando conforme aumenta el orden.

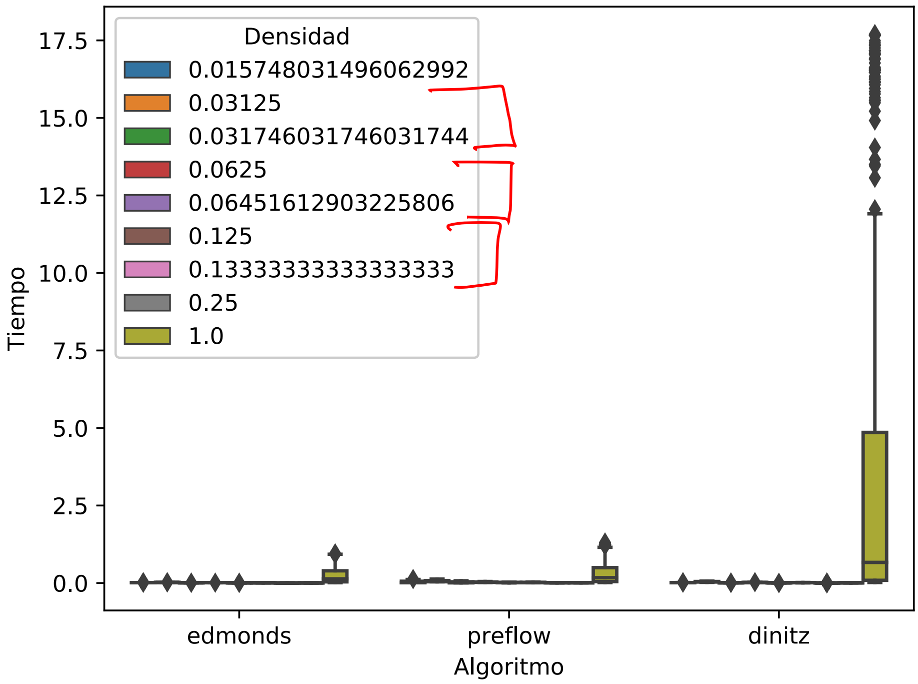


Figura 4: Efecto de la densidad del grafo en el tiempo de ejecución (segundos).

En la figura 4 se puede observar como los tiempos de ejecución conforme la variación es muy parecida, pero cuando la densidad es 1, los tiempos de ejecución aumentan.

4. Conclusiones

Se realizó un análisis de varianza, los resultados se muestran en el cuadro 1, donde con los $p-valores$ se puede concluir si existe efecto entre los factores.

Como el $p-valor$ es mayor que 0.05, para el efecto del generador, densidad y orden respecto al tiempo se puede concluir que existe una relación entre cada uno de los factores mencionados con el tiempo.
Mientras que para el algoritmo el $p-valor$ es menor que 0.05, es decir que el tipo de algoritmo elegidos no muestra una diferencia con respecto al tiempo, no existe una relación.

n - - n

- - -

< 0.001

≈ 0

	sum sq	df	F	PR(>F)	
Generador	-3.845372e-07	2.0	-1.200057e-07	1.000000e+00	1
Algoritmo	7.937227e+02	2.0	2.477036e+02	1.199815e-95	≈ 0
Generador:Algoritmo	9.426795e+02	4.0	1.470949e+02	8.763968e-109	≈ 0
Generador:Orden	9.029616e-02	2.0	2.817947e-02	9.722143e-01	0.97
Orden:Algoritmo	2.768004e+03	2.0	8.638337e+02	5.330766e-263	≈ 0
Orden	8.167572e-10	1.0	5.097843e-10	9.999820e-01	
Densidad	-7.522133e-08	1.0	-4.694988e-08	1.000000e+00	1
Generador:Densidad	9.270394e-02	2.0	2.893088e-02	9.714841e-01	
Algoritmo:Densidad	1.198214e+03	2.0	3.739365e+02	2.499255e-136	≈ 0
Orden:Densidad	9.131569e-02	1.0	5.699528e-02	8.113371e-01	
Residual	2.855053e+03	1782.0			

Cuadro 1: Resultados del ANOVA.

Referencias

[1] Schaeffer E. *Optimización de flujo en redes, 2019.*
`https://elisa.dyndns-web.com/teaching/opt/flow/`

[2] Kamada T. *An algorithm for Drawing General Undirected Graphs*
`Information Processing Letters, 1988`

[3] Moreno A. *Repositorio de flujo en redes, 2019*
`https://github.com/angisabel44`

Tarea 4

Al realizar el reporte de la actividad indicada, se obtuvo retroalimentación, de la cual se hicieron las siguientes correcciones: se corrigió la ortografía, se corrigieron las imágenes en python, se hicieron correcciones en el cuadro del ANOVA.

Tarea 4: Complejidad asintótica experimental

5171

2 de junio de 2019

Objetivo: Utilizando los generadores de grafos de NetworkX, se selecciona por lo menos tres métodos de generación de grafos. Con cada generador, se generan cuatro diferentes órdenes en escala logarítmica (16, 32, 64, 128) y 10 grafos distintos de cada orden. Se le asigna pesos no-negativos normalmente distribuidos a las aristas para que se puedan utilizar como instancias del problema de flujo máximo.
Eligiendo por lo menos tres implementaciones de NetworkX de los algoritmos de flujo máximo, se ejecuta los algoritmos seleccionados con cinco diferentes pares de fuente-sumidero. Con métodos estadísticos y visualizaciones científicas se determina:

- el efecto que el generador de grafo usado tiene en el tiempo de ejecución.
- el efecto que el algoritmo usado tiene en el tiempo de ejecución.
- el efecto que el orden del grafo tiene en el tiempo de ejecución.
- el efecto que la densidad del grafo tiene en el tiempo de ejecución.

1. Generadores implementados

A continuación se describe los 3 métodos de generación de grafos.

1.1. Complete

Devuelve el grafo completo con n nodos.

1.2. Wheel

Devuelve el grafo: un solo nodo central conectado a cada nodo del gráfico de ciclo del nodo $(n-1)$. Las etiquetas de nodo son los números enteros de 0 a $n-1$.

1.3. Cycle

Devuelve el grafo de ciclo C_n sobre n nodos. C_n es la ruta n con dos nodos finales conectados.

2. Algoritmos implementados

A continuación se describe los algoritmos de flujo máximo implementados, que fueron elegidos por el menor tiempo de ejecución.

2.1. Edmons

Encuentra un flujo máximo de un solo producto utilizando el algoritmo Edmonds-Karp. Esta función devuelve la red residual resultante después de calcular el flujo máximo.

Este algoritmo tiene un tiempo de ejecución de $O(nm^2)$ para n nodos y m aristas.

2.2. Preflow

Encuentra un flujo máximo de un solo producto utilizando el algoritmo de empuje previo al flujo de la etiqueta más alta. Este algoritmo tiene un tiempo de ejecución de O $(n^2\sqrt{m})$ para los nodos n y m aristas.

2.3. Dinitz

Encuentra un flujo máximo de un solo producto utilizando el algoritmo de Dinitz.

3. Resultados

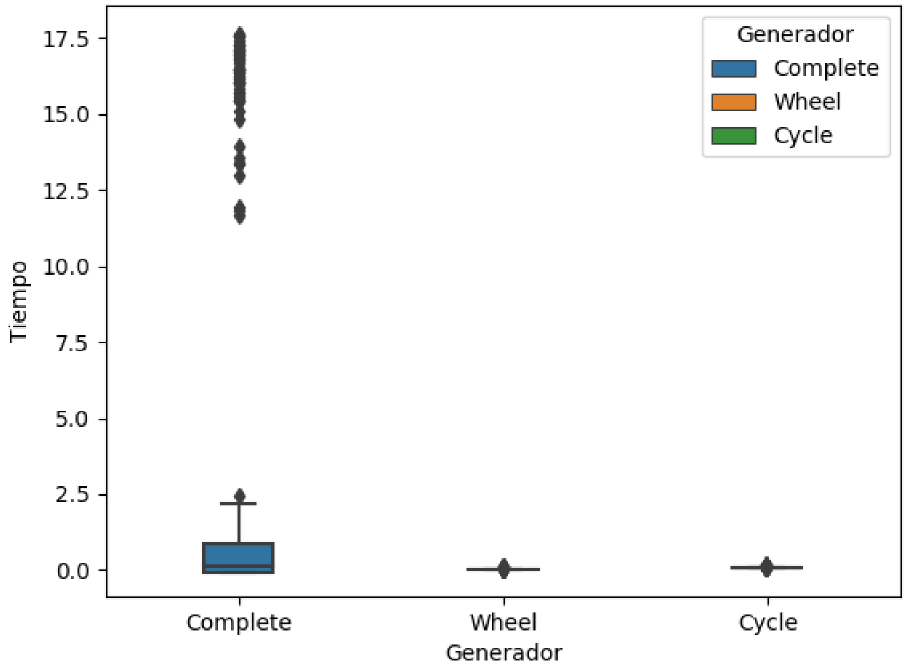


Figura 1: Efecto del generador de grafo en el tiempo de ejecución (segundos).

Se puede observar en la figura 1 que el generador que toma más tiempo en su ejecución es el generador Complete, mientras que en los otros dos sus tiempos son muy parecidos.—

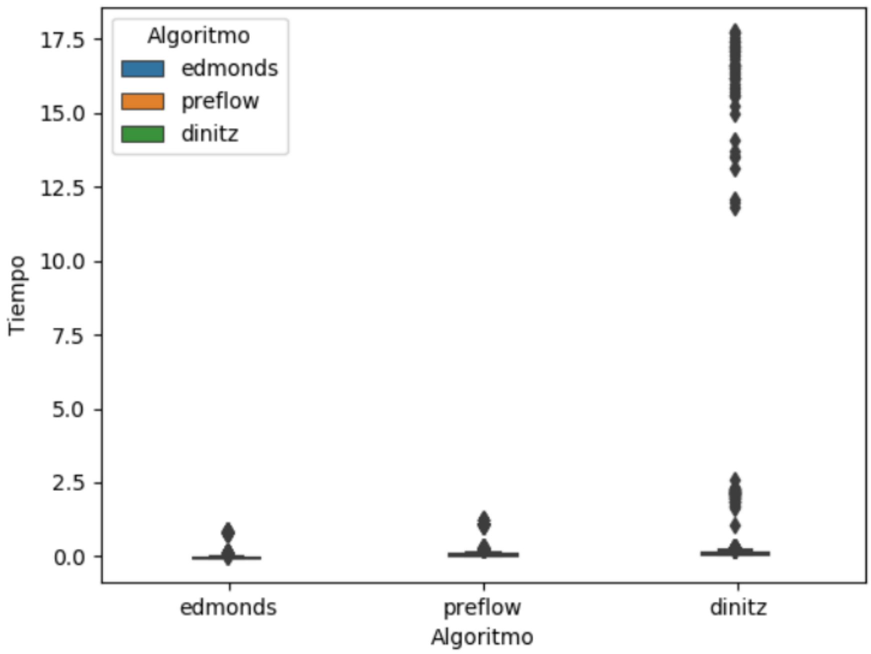


Figura 2: Efecto del algoritmo en el tiempo de ejecución (segundos).

En la figura 2 los tiempos de ejecución de los algoritmos son muy parecidos, por lo que se tendría que realizar un análisis estadístico para concluir del posible efecto.

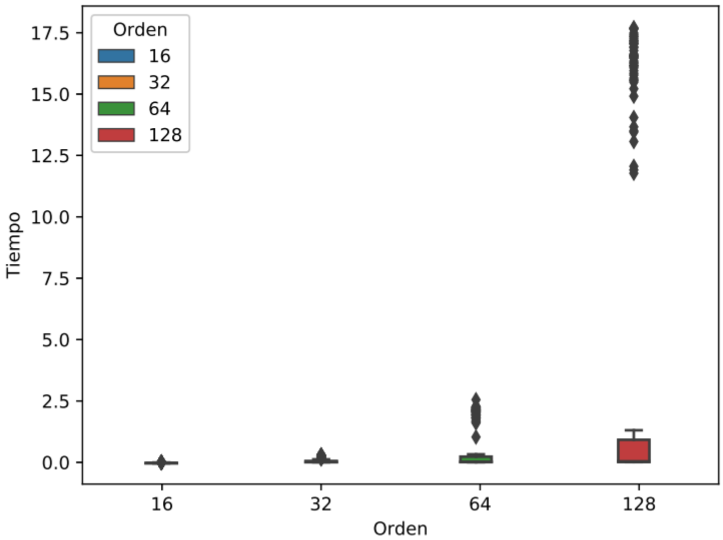


Figura 3: Efecto del orden del grafo en el tiempo de ejecución (segundos).

En la figura 3 se puede observar como ligeramente los tiempos van aumentando conforme aumenta el orden.

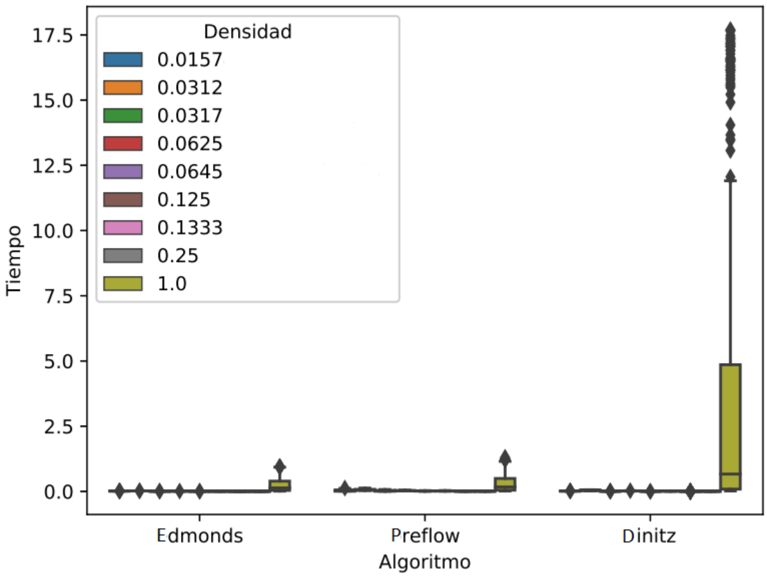


Figura 4: Efecto de la densidad del grafo en el tiempo de ejecución (segundos).

En la figura 4 se puede observar como los tiempos de ejecución conforme la variación es muy parecida, pero cuando la densidad es 1, los tiempos de ejecución aumentan.

	sum sq	df	F	PR(>F)
Generador	-3.84e-07	2	-1.2e-07	≈ 1
Algoritmo	793	2	247	≈ 0
Generador:Algoritmo	942	4	1.47e+02	≈ 0
Generador:Orden	0.0902	2	0.028	$\approx 0{,}97$
Orden:Algoritmo	2768	2	863	≈ 0
Orden	≈ 0	1	≈ 0	$\approx 0{,}99$
Densidad	-7.522e-08	1	-4.694e-08	≈ 1
Generador:Densidad	0.0927	2	0.0289	$\approx 0{,}97$
Algoritmo:Densidad	1198.21	2	373.93	≈ 0
Orden:Densidad	0.0913	1	0.0569	$\approx 0{,}8$
Residual	2855.05	1782		

Cuadro 1: Resultados del ANOVA.

4. Conclusiones

Se realizó un análisis de varianza, los resultados se muestran en el cuadro 1, donde con los p-valores se puede concluir si existe efecto entre los factores.

Como el p-valor es mayor que 0.05, para el efecto del generador, densidad y orden respecto al tiempo se puede concluir que existe una relación entre cada uno de los factores mencionados con el tiempo.
Mientras que para el algoritmo el p-valor es menor que 0.05, es decir que el tipo de algoritmo elegidos no muestra una diferencia con respecto al tiempo, no existe una relación.

Referencias

[1] SCHAEFFER E. *Optimización de flujo en redes, 2019.*
https://elisa.dyndns-web.com/teaching/opt/flow/

[2] KAMADA T. *An algorithm for Drawing General Undirected Graphs*
Information Processing Letters, 1988

[3] MORENO A. *Repositorio de flujo en redes, 2019*
https://github.com/angisabel44

Tarea 5

Se realizó por completo la actividad 5.

Tarea 5: Caracterización Estructural de las instancias

5171

4 de junio de 2019

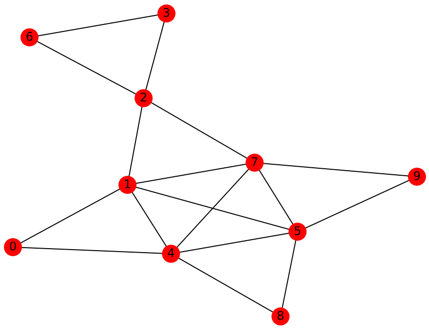
Objetivo: Se selecciona un generador de grafo y el más eficiente de los algoritmos de la tarea anterior, se visualiza por lo menos cinco de las instancias producidas por el generador seleccionado y se visualiza esos grafos con un algoritmo de acomodo que parece producir el resultado más entendible. Se calcula las siguientes características estructurales para todos sus vértices:

- distribución de grado (inglés: degree distribution)
- coeficiente de agrupamiento (inglés: clustering coefficient)
- centralidad de cercanía (inglés: closeness centrality),
- centralidad de carga (inglés: load centrality),
- excentricidad (inglés: eccentricity)
- PageRank.

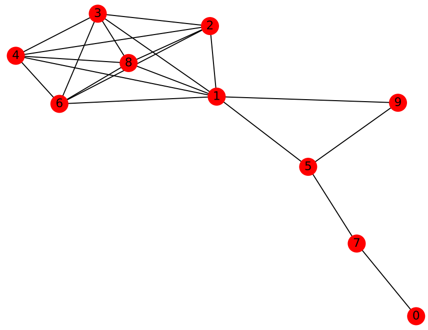
Además se diseña, se ejecuta y se analiza un experimento que busca establecer si estas características de los vértices afectan a tiempo de ejecución del algoritmo seleccionado o al valor del óptimo de alguna manera sistemática, buscando concluir cuáles vértices son buenas fuentes, cuáles son buenos sumideros, y cuáles sería mejor no usar como ninguno si uno busca obtener un alto flujo y no batallar demasiado con el tiempo de ejecución del algoritmo.

Generador usado

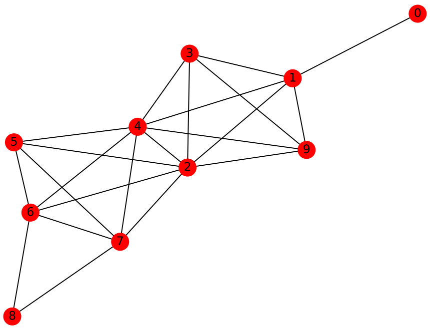
El generador usado es geométrico aleatorio, ya que se encontró aplicación para la configuración de redes inalámbricas. El modelo de grafo geométrico aleatorio coloca los nodos uniformemente al azar en el cubo unitario. Dos nodos se unen en un borde si la distancia euclidiana está entre los nodos. En la figura 1 se observa los grafos obtenidos con este generador.



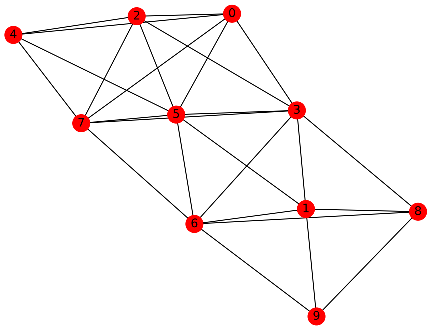
(a) grafo 1



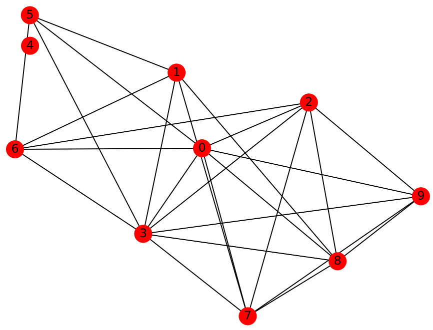
(b) grafo 2



(c) grafo 3



(d) grafo 4

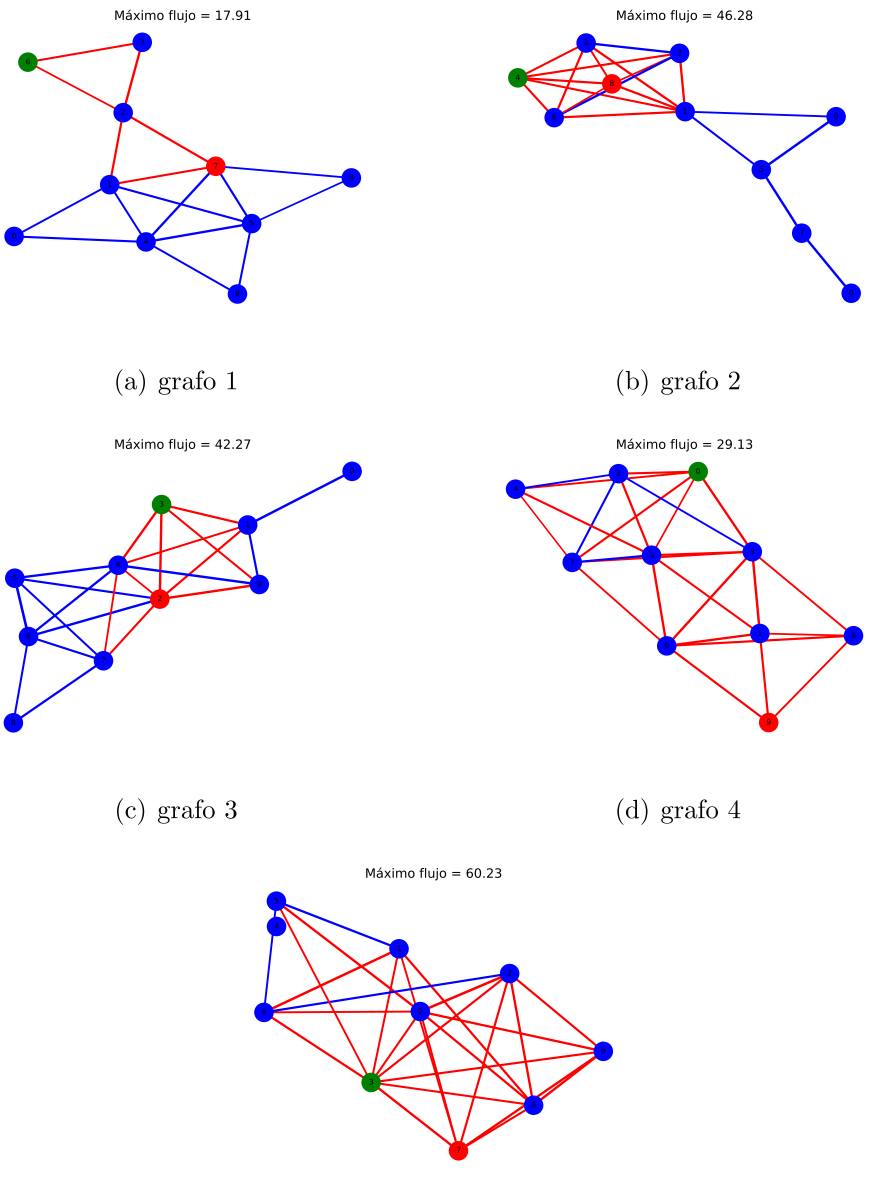


(e) grafo 5

Figura 1: Grafos generados

Nodos fuente y sumidero

En la Figura 2 el nodo verde indica que es un nodo fuente y el nodo rojo indica que es un nodo sumidero, se puede observar cuales de los nodos es mejor usar como fuente y sumidero, dado que arroja el mayor valor de flujo máximo.



(a) grafo 1

(b) grafo 2

(c) grafo 3

(d) grafo 4

(e) grafo 5

Figura 2: Visualización del nodo fuente y sumidero para cada grafo

Además se midió el tiempo para cada instancia, tomando en cuenta que no solamente los mejores fuentes y sumideros fueran por el valor de su flujo máximo, sino también por el menor tiempo de ejecución.
En la figura 3, muestra para cada instancia su tiempo de ejecución del cálculo de su flujo máximo.

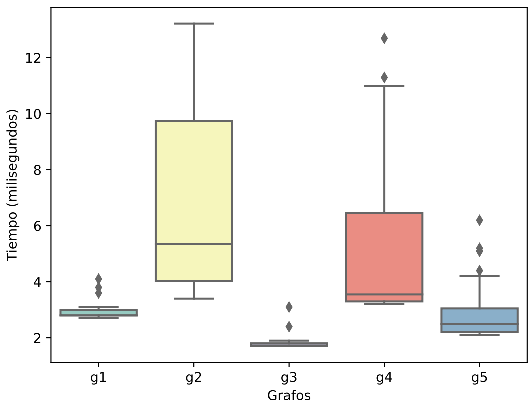
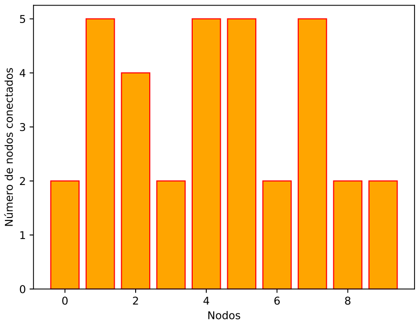


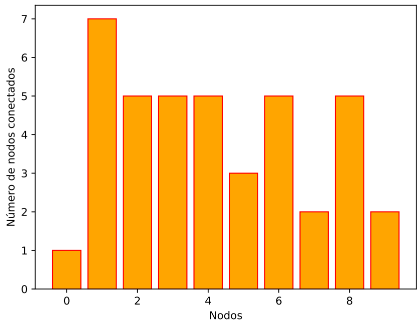
Figura 3: Tiempo de ejecución

Distribución de grado

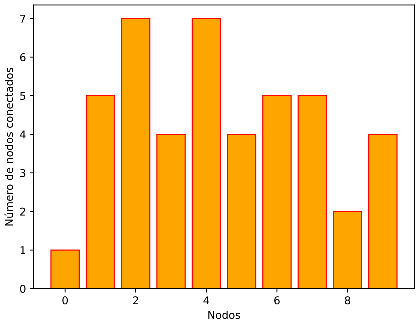
Es el número de conexiones que tiene con otros nodos y la distribución de grados es la distribución de probabilidad de estos grados en todo el grafo.



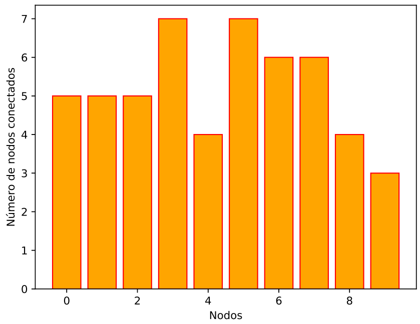
(a) grafo 1



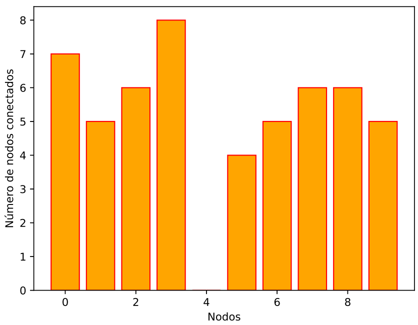
(b) grafo 2



(c) grafo 3



(d) grafo 4

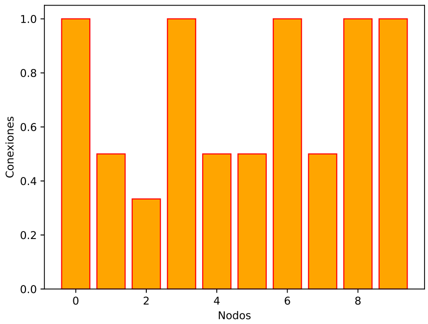


(e) grafo 5

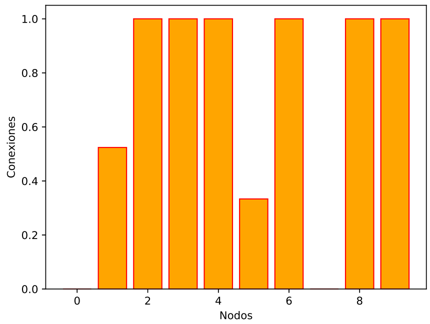
Figura 4: Histograma de distribución de grado para cada grafo.

Coeficiente de agrupamiento

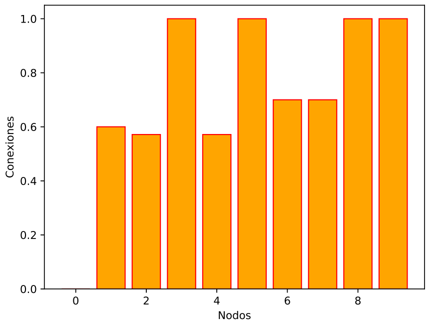
Cuantifica qué tanto está de agrupado (o interconectado) con sus vecinos.



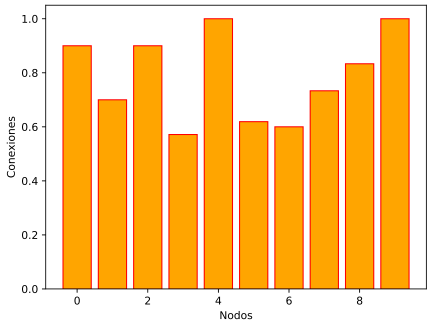
(a) grafo 1



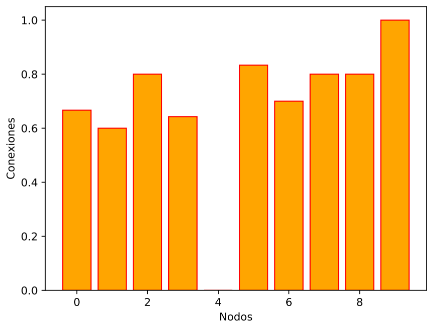
(b) grafo 2



(c) grafo 3



(d) grafo 4

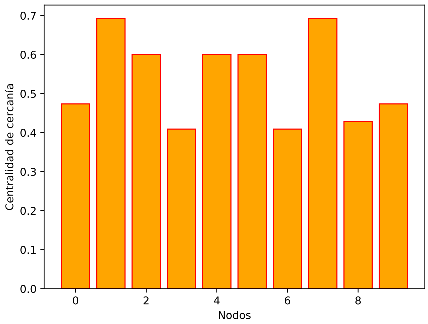


(e) grafo 5

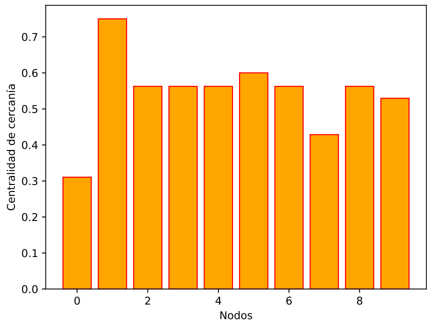
Figura 5: Histograma de agrupamiento para cada grafo.

Centralidad de cercanía

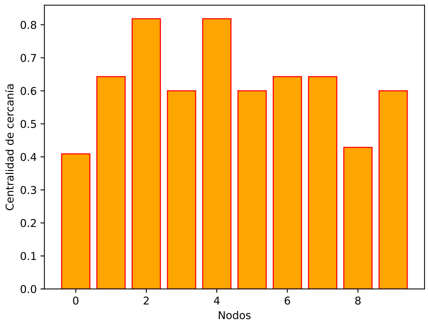
El promedio de las distancias del vértice a todos los demás.



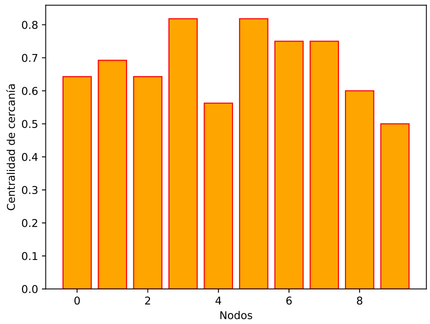
(a) grafo 1



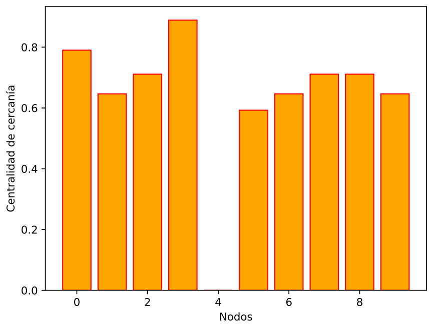
(b) grafo 2



(c) grafo 3



(d) grafo 4

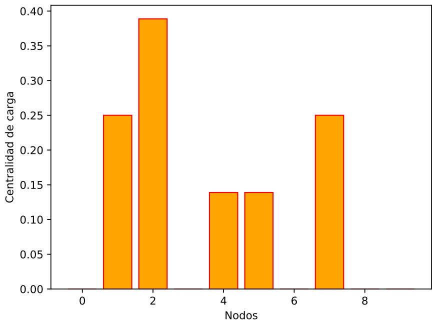


(e) grafo 5

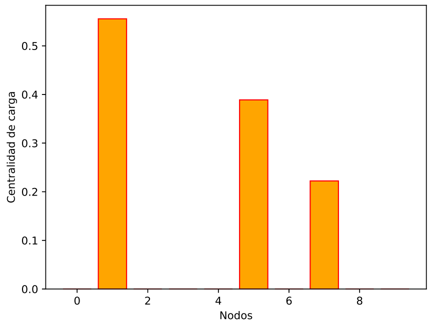
Figura 6: Histograma de centralidad de cercanía para cada grafo.

Centralidad de carga

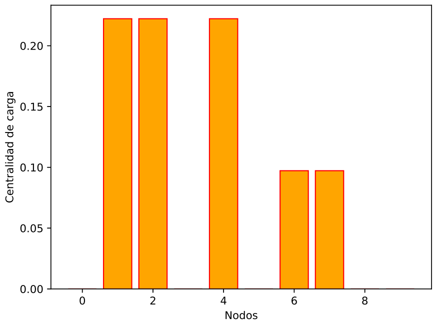
Es la fracción de todas las rutas más cortas que pasan a través de ese nodo.



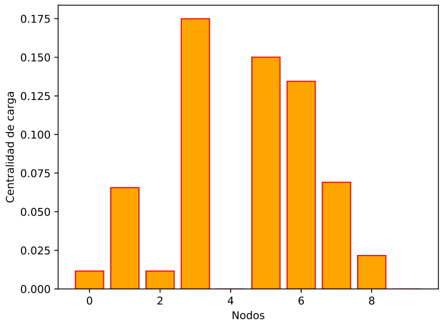
(a) grafo 1



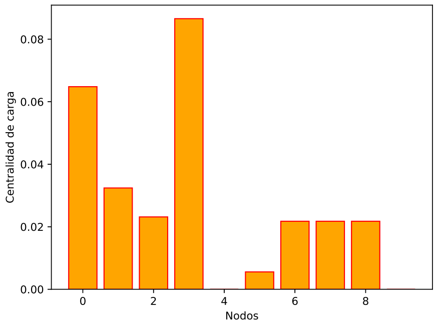
(b) grafo 2



(c) grafo 3



(d) grafo 4

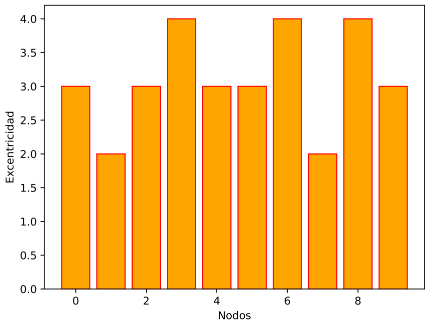


(e) grafo 5

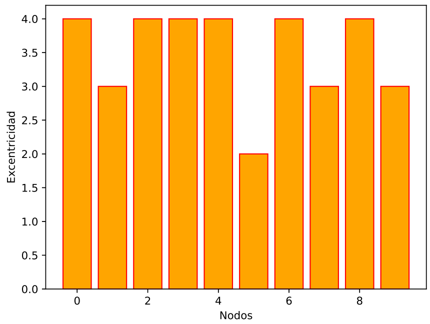
Figura 7: Histograma de centralidad de carga para cada grafo.

Excentricidad

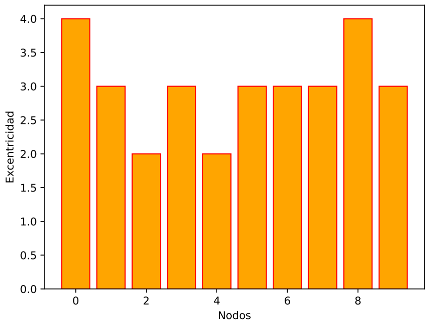
La excentricidad de un nodo v es la maxima distancia de v a todos los nodos en G.



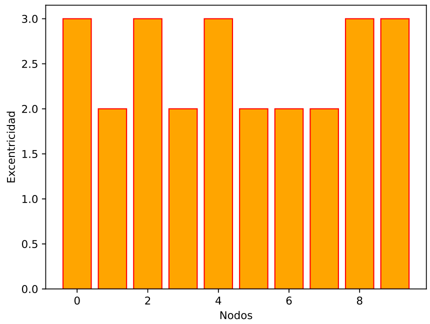
(a) grafo 1



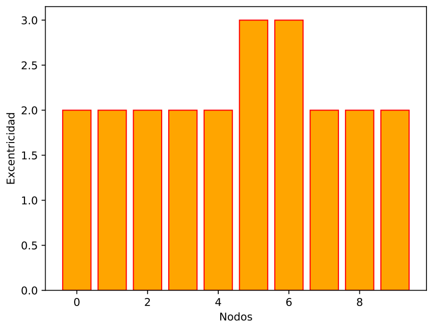
(b) grafo 2



(c) grafo 3



(d) grafo 4

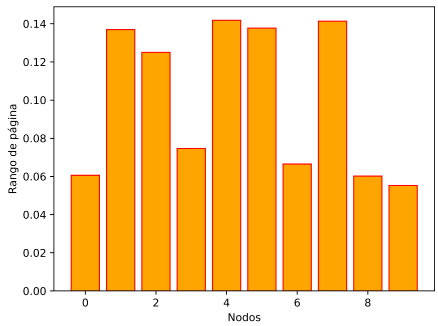


(e) grafo 5

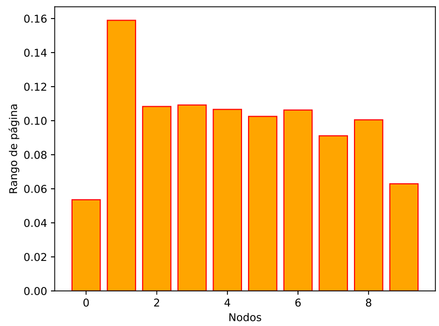
Figura 8: Histograma de excentricidad cada grafo.

Rango de página

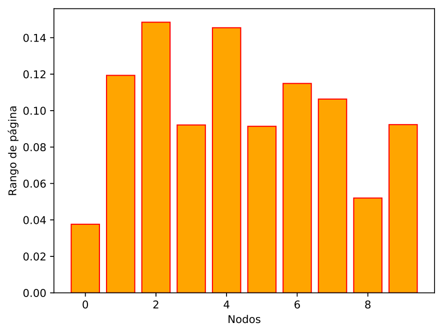
Calcula una clasificación de los nodos en el gráfico G en función de la estructura de los enlaces entrantes. Originalmente fue diseñado como un algoritmo para clasificar páginas web.



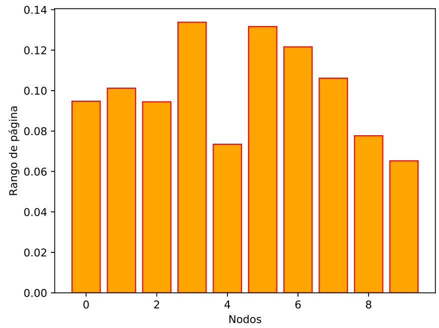
(a) grafo 1



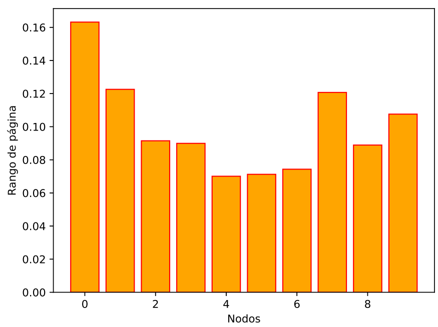
(b) grafo 2



(c) grafo 3



(d) grafo 4



(e) grafo 5

Figura 9: Histograma de rango de página para cada grafo.

Resultados

Se realizó un análisis de las relaciones de las características estudiadas contra el tiempo, de los cuales se presentan los resultados en el cuadro 1. Se puede observar que solamente presentan relación entre las características:

- Distribución de grado y Excentricidad
- Distribución de grado y Coeficiente de Agrupamiento

Además no presentan relación respecto al tiempo.

	coef	std err	t	$P > \|t\|$	[0.025	0.975]
Intercept	0.1072	0.334	0.321	0.752	-0.594	0.809
DisGrado	-0.0234	0.069	-0.340	0.738	-0.168	0.121
CoefAgrup	-0.0149	0.289	-0.052	0.959	-0.622	0.592
CentralidadCerc	-0.0013	0.478	-0.003	0.998	-1.005	1.002
CentralidadCar	-1.2646	2.075	-0.609	0.550	-5.625	3.096
Excentricidad	-0.0492	0.079	-0.621	0.543	-0.216	0.117
PageRank	3.0843	5.919	0.521	0.609	-9.350	15.519
DisGrado:CoefAgrup	-0.0074	0.041	-0.182	0.858	-0.093	0.078
DisGrado:CentralidadCerc	0.0735	0.037	1.990	0.062	-0.004	0.151
DisGrado:CentralidadCar	-0.1205	0.143	-0.842	0.411	-0.421	0.180
DisGrado:Excentricidad	0.0006	0.009	0.073	0.943	-0.017	0.019
DisGrado:PageRank	-0.0196	0.301	-0.065	0.949	-0.653	0.613
CoefAgrup:CentralidadCerc	-0.1549	0.255	-0.608	0.551	-0.690	0.380
CoefAgrup:CentralidadCar	0.0912	0.191	0.478	0.638	-0.309	0.492
CoefAgrup:Excentricidad	0.0201	0.056	0.357	0.725	-0.098	0.139
CoefAgrup:PageRank	0.1214	1.439	0.084	0.934	-2.902	3.144
CentralidadCerc:CentralidadCar	1.3754	3.236	0.425	0.676	-5.423	8.174
CentralidadCerc:Excentricidad	0.0572	0.122	0.470	0.644	-0.198	0.313
CentralidadCerc:PageRank	-5.3930	7.274	-0.741	0.468	-20.676	9.890
CentralidadCar:Excentricidad	0.0674	0.262	0.257	0.800	-0.484	0.618
CentralidadCar:PageRank	5.2518	2.348	2.237	0.038	0.319	10.185
Excentricidad:PageRank	-0.1536	0.769	-0.200	0.844	-1.770	1.463

Cuadro 1: Análisis de ANOVA

Referencias

[1] SCHAEFFER E. *Optimización de flujo en redes, 2019.*
https://elisa.dyndns-web.com/teaching/opt/flow/

[2] KAMADA T. *An algorithm for Drawing General Undirected Graphs*
Information Processing Letters, 1988

[3] MORENO A. *Repositorio de flujo en redes, 2019*
https://github.com/angisabel44

Tarea 6:Generalizaciones

5171

4 de junio de 2019

Objetivo: Se describe un problema y su relevancia en el reporte, igual como el estado de arte de algoritmos para ello y sí o no se basa en la resolución del problema de flujo máximo. Se selecciona un generador de instancias, realizando modificaciones,se varia el orden de las instancias de tal forma que pueda estimar experimentalmente la complejidad asintótica del algoritmo creado. Se compara la complejidad observada experimentalmente con un análisis teórico de la complejidad asintótica que su algoritmo tiene, utilizando visualizaciones, cuadros y métodos estadísticos según necesidad para establecer la precisión y confiabilidad de los hallazgos

1. Problema

Supongamos que nos dan un conjunto de n proyectos que podríamos realizar, identificamos cada proyecto por un número entero entre 1 y n. Algunos proyectos no pueden ser iniciados hasta que otros proyectos se completen. Este conjunto de dependencias es descrito por un gráfico acíclico dirigido, donde un arista (i, j) indica que el proyecto i depende del proyecto j. Finalmente, cada proyecto i tiene un beneficio asociado p_i que se nos otorga si el proyecto se completa. Sin embargo, algunos proyectos tienen beneficios negativos, que interpretamos como costos . Podemos optar por terminar cualquier subconjunto X de los proyectos que incluya a todos sus dependientes; Es decir, para cada proyecto $x \in X$, cada proyecto del que depende x también está en X. Nuestro objetivo es encontrar un subconjunto válido de los proyectos cuyo beneficio total es lo más grande posible. En particular, si todos los trabajos tienen un beneficio negativo, la respuesta correcta es no hacer nada. Entonces definimos un grafo G un nodo origen s y nodo destino t en G, donde s tiene un costo o beneficio cero.

2. Estado del arte

La aplicación principal de este tipo de problemas es programación de la producción de open-pit mines, donde se han aplicado diferentes métodos de optimización para resolverlos. Entre ellos se encuentran:

- Dagdelen and Johnson (1986)
- Caccetta and Hill (2003)
- Ramazan (2007)

- Gershon, (1987)
- Tolwinski and Underwood, 1996

por mencionar algunos.

3. Instancias generadas

Se utilizo el algoritmo de acomodo spectral, para mejor visualización de los grafos, en la figura 1 se muestra un ejemplo de una instancia, con nodo fuente en verde, con nodo sumidero en rojo, ademas se les agregaron pesos.

Referencias

[1] Elisa S. *Optimización de flujo en redes, 2019.*
https://elisa.dyndns-web.com/teaching/opt/flow/

[2] Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart *Exploring network structure, dynamics, and function using NetworkX, 2008.*
https://networkx.github.io/documentation/stable/index.html