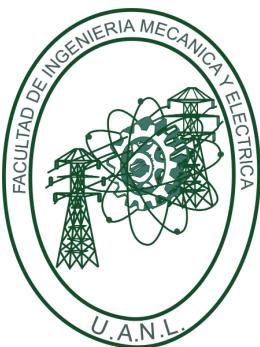


UNIVERSIDAD AUTÓNOMA  
DE NUEVO LEÓN

---

---

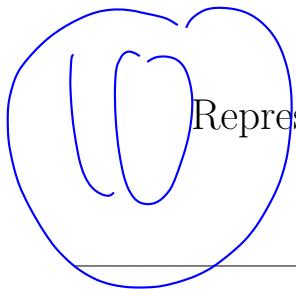
FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA



P O R T A F O L I O

PRESENTA:

Jorge Armando Serna Mendoza  
1985281



# Representación de redes a través de la teoría de grafos

(Serna Mendoza Jorge Armando)  
(Optimización de flujo en redes)

5281

## Resumen

Este trabajo busca representar redes mediante la teoría de grafos, con la implementación de la librería [NetworkX](#) de Python. A cada tipo de grafo se le identifica una aplicación práctica y se representa con un ejemplo inspirado en datos reales.

## Introducción

Un *grafo*  $G = (N, A)$  es una pareja ordenada en la que  $N$  es un conjunto de *nodos* y  $A$  es un conjunto de *arcos*. El conjunto  $A$  se conforma de pares de nodos  $(u, v)$  y se dice que  $u$  y  $v \in N$  son adyacentes. En  $G$  se representa la adyacencia de  $u$  con  $v$ , mediante una línea que une el nodo  $u$  con el nodo  $v$ .

Un *multigrafo* es un grafo que tiene *multiarcos*, es decir, si  $u$  y  $v$  son adyacentes, existe más un arco que une a  $u$  con  $v$ . Por otro lado, un *ciclo* es una sucesión de nodos y arcos tales que forman un camino que comienza y termina en el mismo nodo.

### Grafo simple no dirigido acíclico

vin  
de

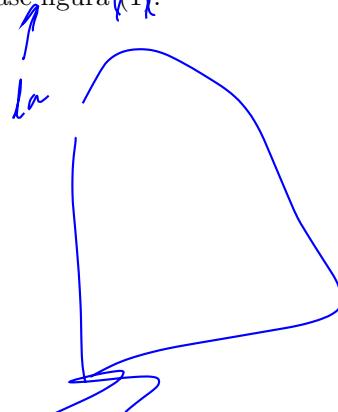
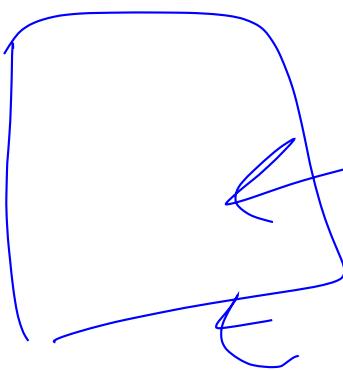
Es un grafo  $G = (N, A)$  que cumple:

- $G$  no tiene multiarcos.
- Los arcos  $(u, v) \in A$  no tienen dirección  $\forall u, v \in N$ .
- $G$  no tiene ciclos.

**Ejemplo:** La Filogenia es la relación de parentesco entre especies, se encarga de estudiar filiación de entidades que derivan unas de otras en un proceso evolutivo. Véase figura 1.

Cuadro 1: Bacterias.

Notación	Nombre
a	Aquifex
b	Thermotoga
c	Planctomyces
d	Cianobacteria
e	Proteobacteria
f	Spirochaetes
g	Cytophaga
h	Gram positiva



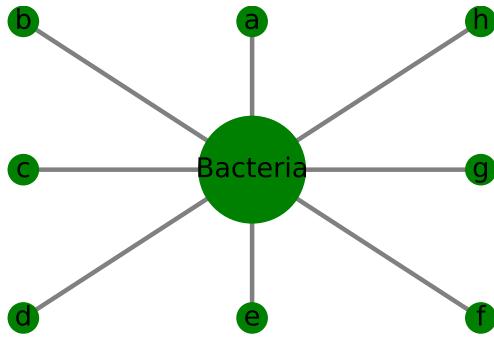


Figura 1: Árbol filogenético de la Bacteria

Código en Python de la figura 1.

```

1 import networkx as nx
2 import matplotlib.pyplot as plt
3
4 #1 grafo simple no dirigido
5 G=nx.Graph()
6 G.add_edges_from([('Bacteria','a'), ('Bacteria','b'), ('Bacteria','c'), ('Bacteria','d'),
7     ),('Bacteria','e'), ('Bacteria','f'), ('Bacteria','g'), ('Bacteria','h')])
8 node1 = {"Bacteria"}
9 node2 = {"a", "b", "c", "d", "e", "f", "g", "h"}
10 pos = {"Bacteria":(0, 0), "a":(0,5), "b":(-5,5), "c":(-5,0), "d":(-5,-5), "e":(0,-5),
11     "f":(5,-5), "g":(5,0), "h":(5,5)}
12 nx.draw_networkx_nodes(G, pos, nodelist=node1, node_size=5000,
13                         node_color='green', node_shape='o', width=5, alpha=1)
14 nx.draw_networkx_nodes(G, pos, nodelist=node2, node_size=400,
15                         node_color='green', node_shape='o', width=5, alpha=1)
16 nx.draw_networkx_edges(G, pos, width=3, alpha=0.5, edge_color='grey')
17 nx.draw_networkx_labels(G, pos, font_size=18)
18 plt.axis('off')
19 plt.savefig('grafo1.eps', format='eps', dpi=1000)

```

## Grafo simple no dirigido cíclico

Es un grafo  $G = (N, A)$  que cumple:

- $G$  no tiene multiarcos.
- Los arcos  $(u, v) \in A$  no tienen dirección  $\forall u, v \in N$ .
- $G$  tiene por lo menos un ciclo.

**Ejemplo:** Travelling Salesman Problem (TSP) es un problema que consiste en dada una lista de ciudades ( $n$ ) y las distancias entre cada par de ellas (aristas), determinar cual es la ruta más corta posible que visita cada ciudad una vez y al finalizar regresa a la ciudad origen. Véase figura (2).

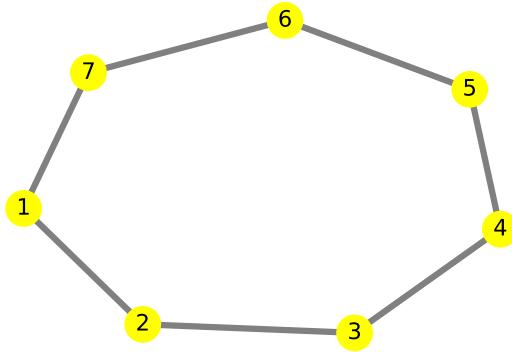


Figura 2: TSP con  $n=7$

### Grafo simple no dirigido reflexivo

Es un grafo  $G = (N, A)$  que cumple:

- $G$  no tiene multiarcos.
- Los arcos  $(u, v) \in A$  no tienen dirección  $\forall u, v \in N$ .
- $G$  tiene por lo menos un nodo  $u$  tal que la relación  $(u, u) \in A$ .

**Ejemplo:** (Producto Cartesiano). Sea  $A = \{1, 2, 3, 4, 5, 6\}$  un conjunto con 6 elementos y sea  $A_1 = \{1, 2, 3, 4, 5\}$  un subconjunto de  $A$ . Se define el producto cartesiano  $B = A \times A_1$ .

$$B = \left\{ \begin{array}{ccccccc} (1, 1) & (2, 1) & (3, 1) & (4, 1) & (5, 1) & (6, 1) \\ (1, 2) & (2, 2) & (3, 2) & (4, 2) & (5, 2) & (6, 2) \\ (1, 3) & (2, 3) & (3, 3) & (4, 3) & (5, 3) & (6, 3) \\ (1, 4) & (2, 4) & (3, 4) & (4, 4) & (5, 4) & (6, 4) \\ (1, 5) & (2, 5) & (3, 5) & (4, 5) & (5, 5) & (6, 5) \end{array} \right\}$$

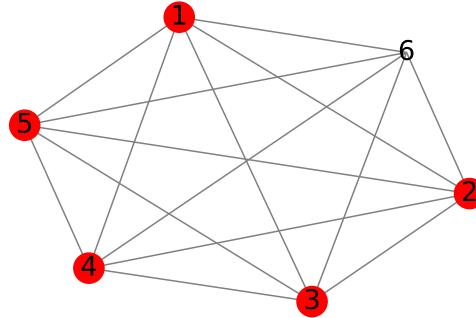


Figura 3: Representación de  $B$  como un grafo

En la figura 3 se representa de color rojo a los nodos de  $G$  que cumplen tener en  $A$  arcos de la forma  $(u, u)$ .

## Grafo simple dirigido acíclico

Es un grafo  $G = (N, A)$  que cumple:

- $G$  no tiene multiarcos.
- Existe al menos un arco  $(u, v) \in A$  con dirección.
- $G$  no tiene ciclos.

**Ejemplo:** (Árbol genealógico). Un árbol genealógico visto como un grafo, es una representación de los descendientes de un individuo de forma organizada. Véase figura (4).

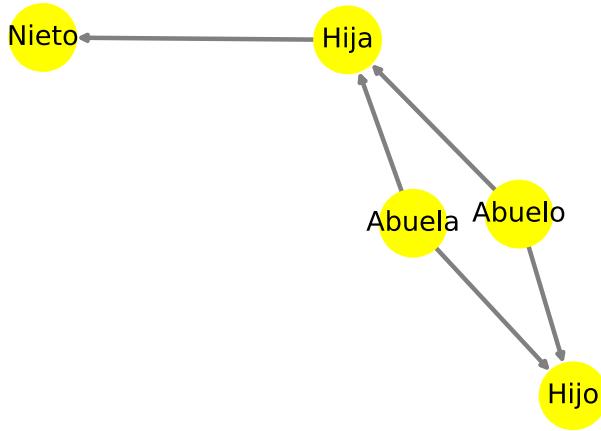


Figura 4: Ejemplo de un árbol genealógico

## Grafo simple dirigido cíclico

Es un grafo  $G = (N, A)$  que cumple:

- $G$  no tiene multiarcos.
- Existe al menos un arco  $(u, v) \in A$  con dirección.
- $G$  tiene al menos un ciclo.

**Ejemplo:** (Ciclo Hidrológico). El ciclo hidrológico es el proceso de circulación del agua entre los distintos compartimentos que forman la hidrosfera, donde intervienen reacciones químicas que provocan cambios de estado físico del agua. Véase figura (5).

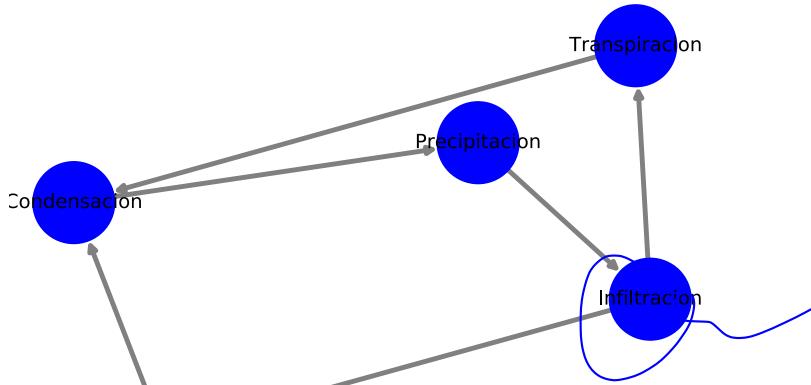


Figura 5: Ejemplo de un árbol genealógico

#ooooff

#ccccff

### Grafo simple dirigido reflexivo

Es un grafo  $G = (N, A)$  que cumple:

- $G$  no tiene multiarcos.
- Existe al menos un arco  $(u, v) \in A$  con dirección.
- $G$  tiene por lo menos un nodo  $u$  tal que la relación  $(u, u) \in A$ .

**Ejemplo:** (Estado del clima). Es la condición en que se encuentra la atmósfera en un determinado momento y lugar. Los estados del clima cambian todos los días con cierta probabilidad. Cuando se representa el clima mediante un grafo, los nodos representan los estados del clima y las aristas la probabilidad de pasar de un estado al otro. Véase figura (6).

Cuadro 2: Estado del clima.

Notación	Clima
1	Lluvioso
2	Soleado
3	Nublado
4	Nevado
5	Frío
6	Ventoso



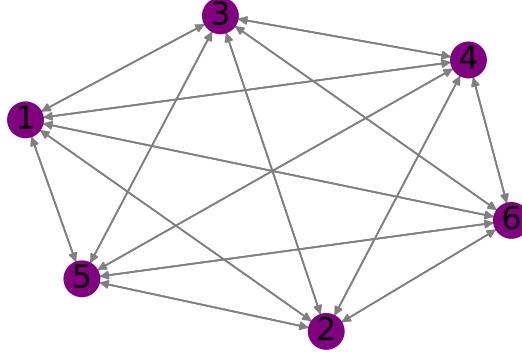


Figura 6: Estados del clima

### Multigrafo no dirigido acíclico

Es un grafo  $G = (N, A)$  que cumple:

- $G$  tiene al menos un par de nodos, tales que tienen más de una arista de adyacencia.
- Los arcos  $(u, v) \in A$  no tienen dirección  $\forall u, v \in N$ .
- $G$  no tiene ciclos.

**Ejemplo:** (Red de rutas). Cuando una red de rutas se representa mediante un grafo, los nodos representan lugares y los arcos representan el camino que une esos lugares. En la figura (7) se considera las rutas de 4 ciudades, tales ciudades pueden tener un único camino (azul) para llegar de ciudad a ciudad o más de un camino (verde) para llegar de ciudad a ciudad.

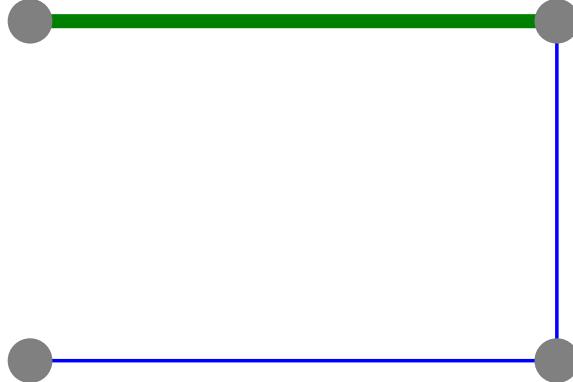


Figura 7: Red de rutas

## Multigrafo no dirigido cíclico

Es un grafo  $G = (N, A)$  que cumple:

- $G$  tiene al menos un par de nodos, tales que tienen mas de una arista de adyacencia.
- Los arcos  $(u, v) \in A$  no tienen dirección  $\forall u, v \in N$ .
- $G$  tiene a lo menos un ciclo.

**Ejemplo:** (Puentes de Königsberg). El problema consistía en encontrar un recorrido para cruzar a pie toda la ciudad, pasando sólo una vez por cada uno de los puentes, y regresando al mismo punto de inicio. En la figura (8) se considera los 4 puntos de inicio posibles, tales puntos pueden tener un puente (azul) para llegar de un punto a otro punto o más de un puente (verde) para llegar de un punto a otro punto.

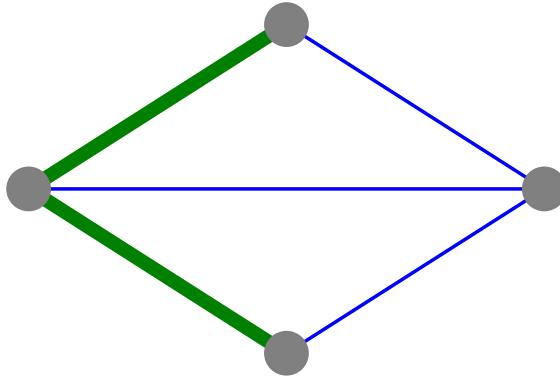


Figura 8: Puentes de Königsberg

## Multigrafo no dirigido reflexivo

Es un grafo  $G = (N, A)$  que cumple:

- $G$  tiene al menos un par de nodos, tales que tienen mas de una arista de adyacencia.
- Los arcos  $(u, v) \in A$  no tienen dirección  $\forall u, v \in N$ .
- $G$  tiene por lo menos un nodo  $u$  tal que la relación  $(u, u) \in A$ .

**Ejemplo:** (Proceso de calidad). En la industria se aplican procesos de calidad a los productos antes de ser sacados al mercado. Cuando un proceso de calidad se representa mediante un grafo, el grafo tiene un nodo origen que representa el inicio del proceso, nodos intermedios que llevan a cabo cada tarea de revisión de calidad, la misma revisión se puede llevar a cabo más de una vez antes de ser pasado a otro proceso de calidad y un nodo destino que representa el fin del proceso.

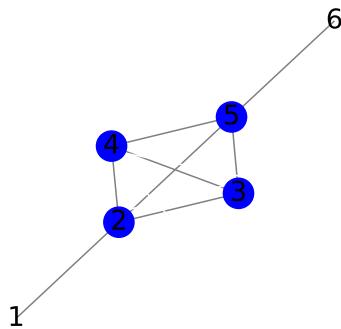


Figura 9: Proceso industrial de calidad

En la figura (9) el nodo 1 representa el inicio del proceso, los nodos intermedios (color azul) representan las actividades de revisión de calidad, las cuales se pueden llevar a cabo mas de una vez y el nodo 6 representa el fin del proceso.

### Multigrafo dirigido acíclico

Es un grafo  $G = (N, A)$  que cumple:

- $G$  tiene al menos un par de nodos, tales que tienen mas de una arista de adyacencia.
- Existe al menos un arco  $(u, v) \in A$  con dirección.
- $G$  no tiene ciclos.

**Ejemplo:** (Red de vuelos en un aeropuerto). En la figura (10) se representa el número de viajes que puede existir entre un aeropuerto de una ciudad y otro de otra ciudad.

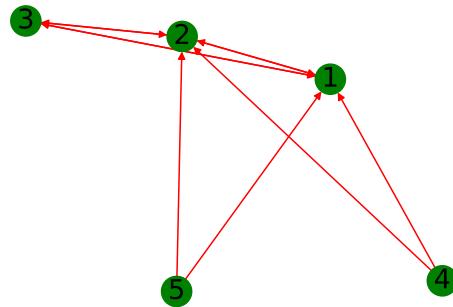


Figura 10: Red de vuelos

## Multigrafo dirigido cíclico

Es un grafo  $G = (N, A)$  que cumple:

- $G$  tiene al menos un par de nodos, tales que tienen mas de una arista de adyacencia.
- Existe al menos un arco  $(u, v) \in A$  con dirección.
- $G$  al menos un ciclo.

**Ejemplo:** (Red de amistad). En la figura (12) se tiene un conjunto de personas que tienen agregado, son amigos, siguen, etc. a otra persona por alguna o algunas páginas de internet como Facebook, Instagram, Twiter, Hi-5, etc.

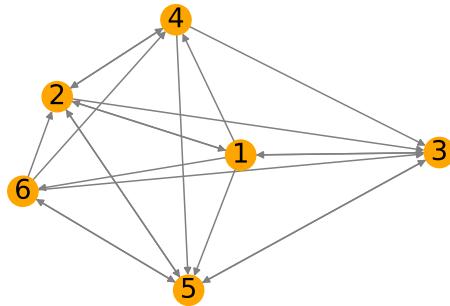


Figura 11: Redes sociales

## Multigrafo dirigido reflexivo

Es un grafo  $G = (N, A)$  que cumple:

- $G$  tiene al menos un par de nodos, tales que tienen mas de una arista de adyacencia.
- Existe al menos un arco  $(u, v) \in A$  con dirección.
- $G$  tiene por lo menos un nodo  $u$  tal que la relación  $(u, u) \in A$ .

**Ejemplo:** (Transmisión de enfermedades). En la figura (12) se representa la red social de un grupo de 6 personas los cuales pueden tener o desarrollar una cierta enfermedad, los arcos en la red representan la probabilidad de ser contagiado.

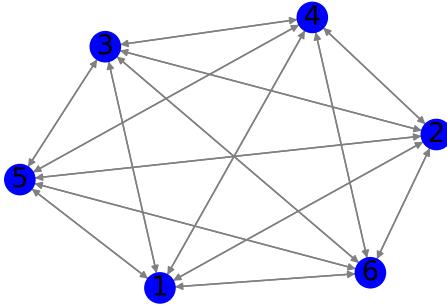


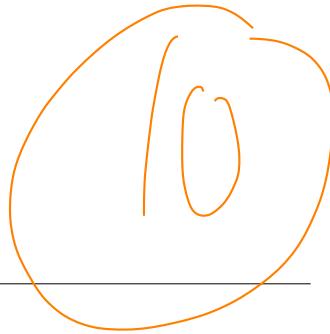
Figura 12: Transmisión de enfermedades

## Referencias

- [1] R. Ahuja, T. Magnanti, and J. Orlin. Network Flows: Theory, Algorithms and Applications. [Prentice Hall], 1993.
- [2] Bazaraa M., Jarvis J., and Sherali H. Linear Programming and Network Flows. Wiley, 4th edition edition, 2010.
- [3] Python. <https://www.python.org/>.
- [4] Schaeffer. <https://elisa.dyndns-web.com/teaching/opt/flow/>.

# Visualización de grafos

Matrícula: 1985281



## Resumen

Este trabajo busca representar grafos con la implementación de las librerías [NetworkX](#) y [matplotlib](#) de Python. Cada grafo de la tarea 1 se dibuja utilizando un diferente algoritmo de acomodo (inglés: [layout](#)).

## Introducción

La librería [NetworkX](#) de Python proporciona algoritmos de posicionamiento para la visualización de grafos. La principal función de los algoritmos es poder personalizar el dibujo del grafo. Además en el cuadro 1 se muestran los algoritmos utilizados en este trabajo y su forma de acomodo.

Cuadro 1: Algoritmos de posicionamiento.

Algoritmo	Posiciona a los nodos:
circular_layout	En un círculo
random_layout	Uniformemente al azar en el cuadrado unitario
spectral_layout	Utiliza los vectores propios Laplacianos
spring_layout	Utiliza el algoritmo <i>Fruchterman-Reingold</i>
shell_layout	En círculos concéntricos
fruchterman_reingold_layout	Utiliza el algoritmo <i>Fruchterman-Reingold</i>
kamada_kawai_layout	Utiliza la función <i>Kamada-Kawai</i>

Estos algoritmos se utilizan para ordenar los doce tipos de grafos de la tarea 1 [5] y a través de las imágenes obtenidas de cada algoritmo se escoge la que sea mejor visualmente para el grafo.

El algoritmo de Fruchterman-Reingold es un algoritmo de diseño dirigido por fuerza. La idea de un algoritmo de diseño dirigido por fuerza es considerar una fuerza entre dos nodos cualquiera y consiste en minimizar la energía del sistema moviendo los nodos y cambiando las fuerzas entre ellos. Por otro lado, la función Kamada-Kawai consiste en la reducción del número de cruces de enlaces en un grafo.

## Grafo simple no dirigido acíclico

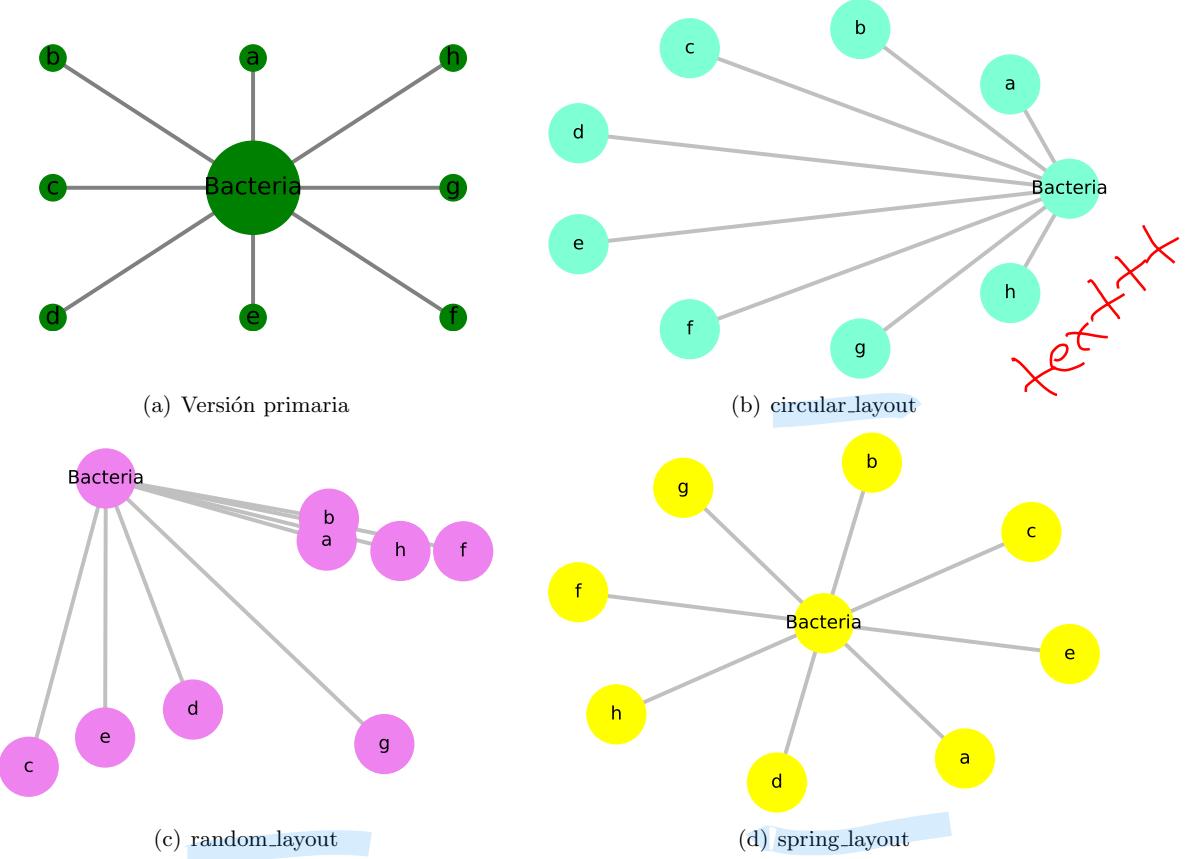


Figura 1: Ejemplo de la Filogenia

En la figura 1 se puede apreciar que el algoritmo de acomodo que mejor dibuja la red, es el algoritmo `spring_layout`, ya que el nodo llamado “Bacteria” que tiene mayor grado y es adyacente al resto de los demás nodos queda el centro. A continuación se muestra parte del código en Python para dibujar grafos con los acomodos `circular_layout`, `random_layout` y `spring_layout`.

```

1 nx.draw(G, with_labels=True, node_size=2000, node_color="aquamarine", pos=nx.circular_layout(G), width=3, edge_color='silver', font_size=14)
2
3 nx.draw(G, with_labels=True, node_size=2000, node_color="violet", pos=nx.random_layout(G), width=3, edge_color='silver', font_size=14)
4
5 nx.draw(G, with_labels=True, node_size=2000, node_color="yellow", pos=nx.spring_layout(G), width=3, edge_color='silver', font_size=14)

```

## Grafo simple no dirigido cíclico

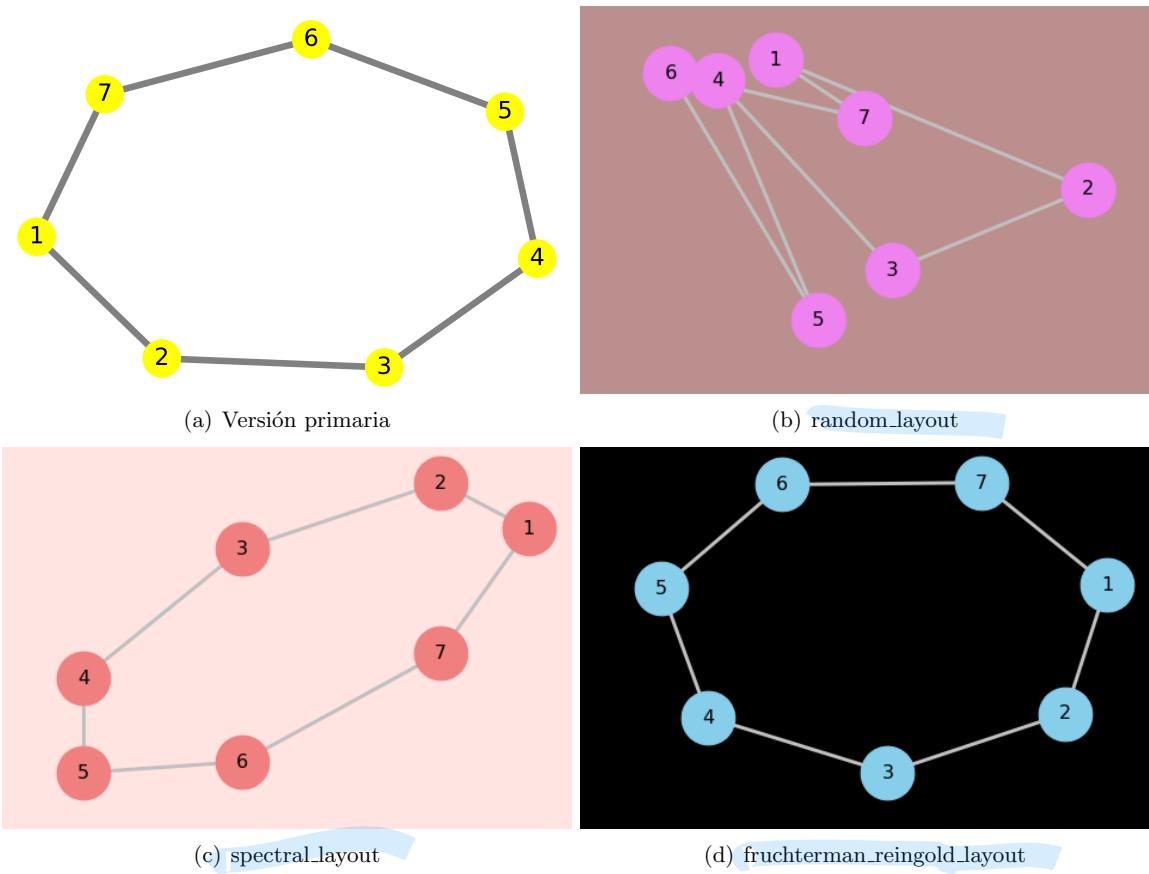


Figura 2: Ejemplo del problema de TSP

En la figura 2 se puede apreciar que el algoritmo de acomodo que mejor dibuja la red es el algoritmo que da por default `NetworkX`, ya que por ejemplo, el algoritmo `random_layout` no genera un dibujo del grafo bueno visualmente. A continuación se muestra parte del código en Python para dibujar grafos encima de una plantilla de algún color.

```

1 fig = plt.figure()
2 nx.draw(G, with_labels=True, node_size=2000, node_color="violet", pos=nx.
3     random_layout(G), width=3, edge_color='silver', font_size=16)
4 fig.set_facecolor("rosybrown")
5
6 fig = plt.figure()
7 nx.draw(G, with_labels=True, node_size=2000, node_color="lightcoral", pos=nx.
8     spectral_layout(G), width=3, edge_color='silver', font_size=16)
9 fig.set_facecolor("mistyrose")
10
11 fig = plt.figure()
12 nx.draw(G, with_labels=True, node_size=2000, node_color="skyblue", pos=nx.
13     fruchterman_reingold_layout(G), width=3, edge_color='silver', font_size=16)
14 fig.set_facecolor("black")

```

## Grafo simple no dirigido reflexivo

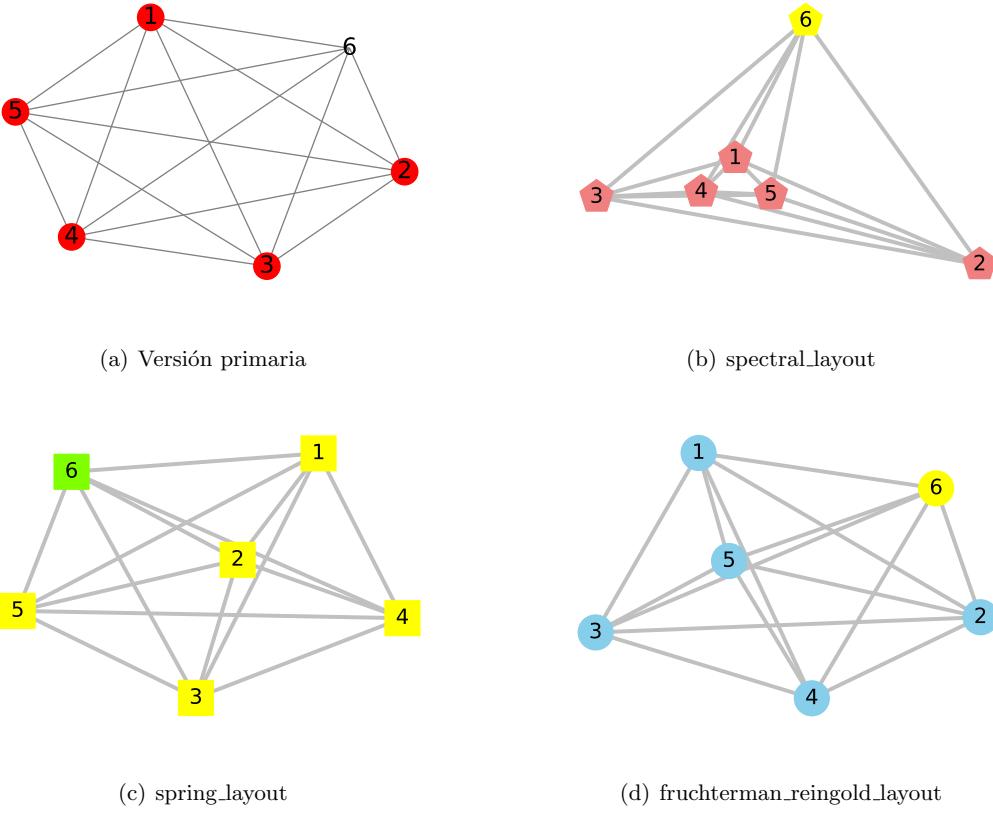


Figura 3: Ejemplo de Producto Cartesiano

En la figura 3 se puede apreciar que el algoritmo de acomodo que mejor dibuja la red es el algoritmo que da por default `NetworkX`, ya que por ejemplo, el algoritmo `spectral_layout` no genera un dibujo del grafo bueno visualmente. A continuación se muestra parte del código en Python para dibujar grafos con nodos de distinta forma y distinto color.

```

1 nx.draw_networkx (G, with_labels=True, node_size=700, node_color = color_map, pos=nx.
    spectral_layout(G), node_shape='p', width=3, edge_color='silver', font_size=16)
2
3 nx.draw_networkx (G, with_labels=True, node_size=700, node_color = color_map, pos=nx.
    spring_layout(G), node_shape='s', width=3, edge_color='silver', font_size=16)
4
5 nx.draw_networkx (G, with_labels=True, node_size=700, node_color = color_map, pos=nx.
    fruchterman_reingold_layout(G), node_shape='o', width=3, edge_color='silver',
    font_size=16)

```

## Grafo simple dirigido acíclico

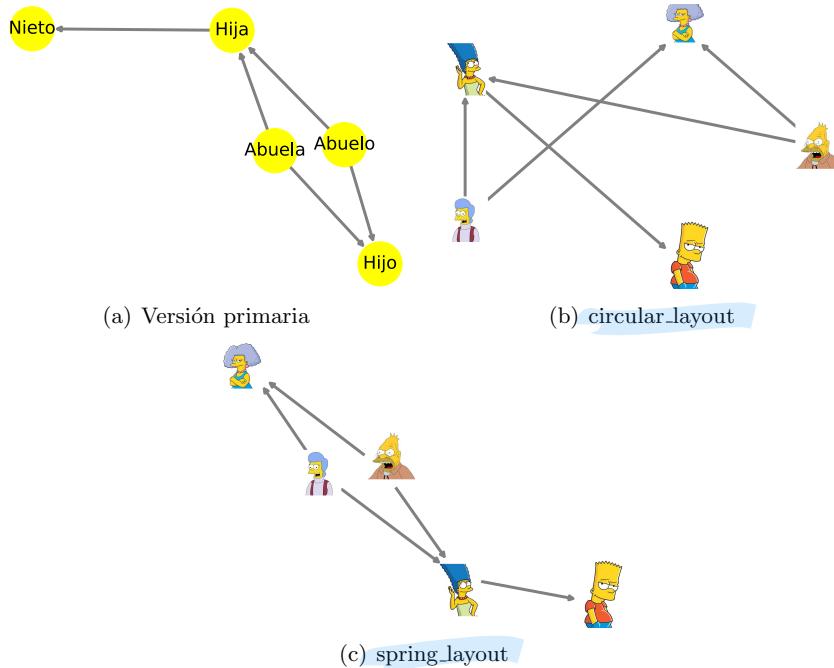


Figura 4: Ejemplo de árbol genealógico

En la figura 4 se puede apreciar que el algoritmo de acomodo que mejor dibuja la red es el algoritmo que da por default `NetworkX`, ya que por ejemplo, el algoritmo `circular_layout` no genera un dibujo del grafo bueno visualmente. A continuación se muestra parte del código en Python para dibujar grafos con nodos que tienen imágenes encima, basado en un código ejemplo [8].

```

1 from PIL import Image
2 def main() :
3     try :
4         img = Image.open("grafo4.png")
5         img1 = Image.open("abuelo.png")
6         img2 = Image.open("abuela.png")
7         img3 = Image.open("mama.png")
8         img4 = Image.open("hermana.png")
9         img5 = Image.open("hijo.png")
10        img.paste(img1, (2200,1250))
11        img.paste(img2, (1200,1500))
12        img.paste(img3, (3200,3100))
13        img.paste(img4, (100,0))
14        img.paste(img5, (5200,3000))
15    except IOError :
16        pass
17 if __name__ == "__main__":
18     main()

```

## Grafo simple dirigido cíclico

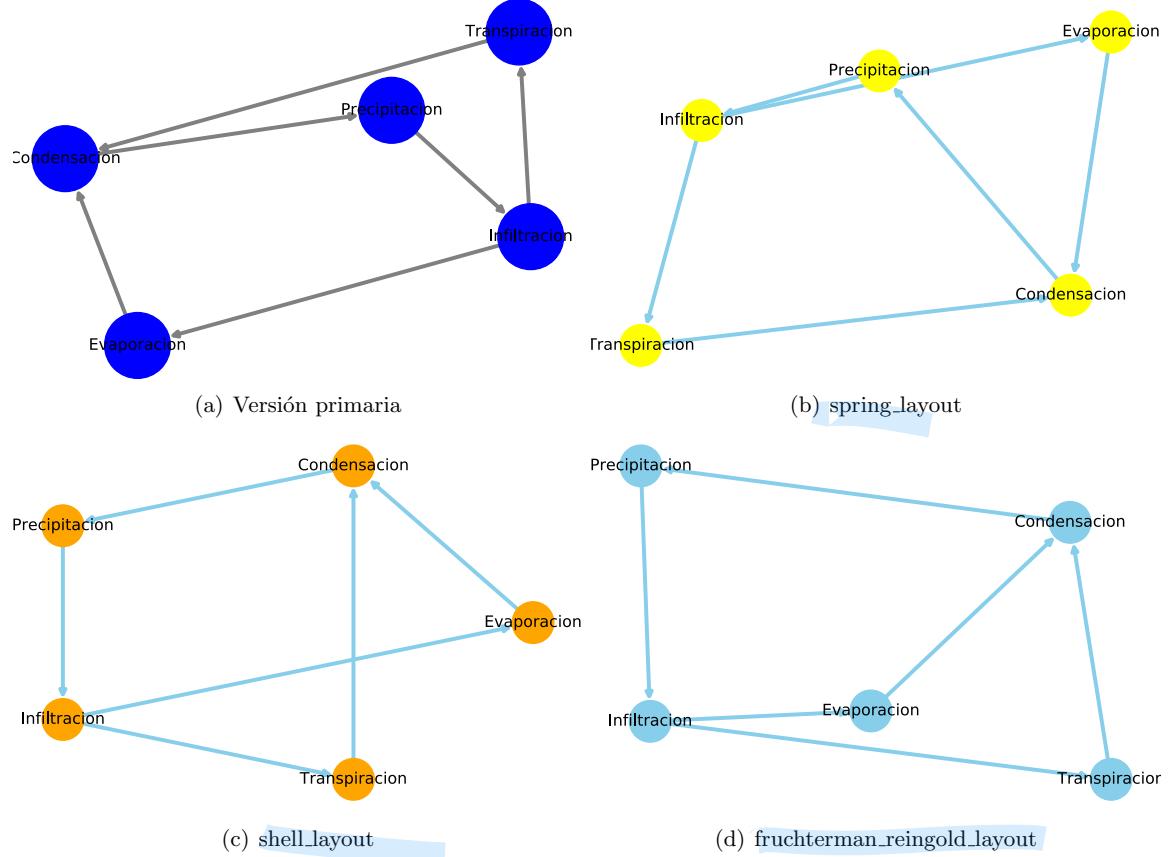


Figura 5: Ejemplo del ciclo del agua

En la figura 5 se puede apreciar que el algoritmo de acomodo que mejor dibuja la red es el algoritmo que da por default `fruchterman_reingold_layout` ya que por ejemplo, el algoritmo `spring_layout` no genera un dibujo del grafo bueno visualmente.

## Grafo simple dirigido reflexivo

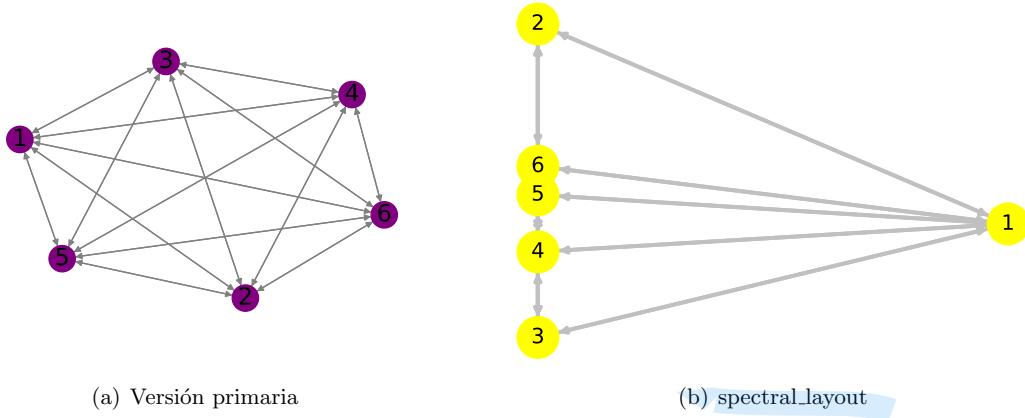


Figura 6: Ejemplo de estados del clima

En la figura 6 se puede apreciar que los algoritmos de acomodo que mejor dibujan la red es el algoritmo que da por default NetworkX y el algoritmo de acomodo `spectral_layout`, ya que ambos algoritmos generan dibujos del gráfico buenos visualmente.

## Multigrafo no dirigido acíclico

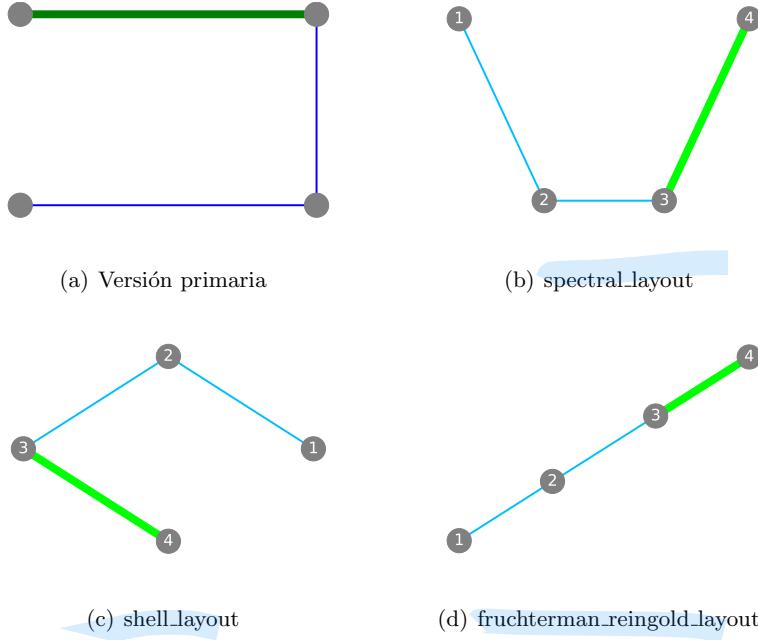


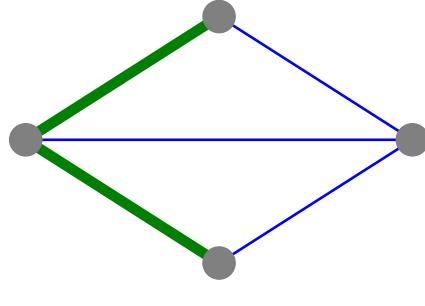
Figura 7: Ejemplo de red de rutas

En la figura 7 se puede apreciar que todos los algoritmos dibujan visualmente bien el grafo, sin embargo esto es por que el ejemplo es pequeño. Por lo que para este caso no se puede decir con certeza que algoritmo es mejor visualmente que otro para dibujar el grafo del ejemplo de red de rutas. A continuación se muestra parte del código en Python para dibujar grafos que tengan aristas con distinto grosor y distinto color.

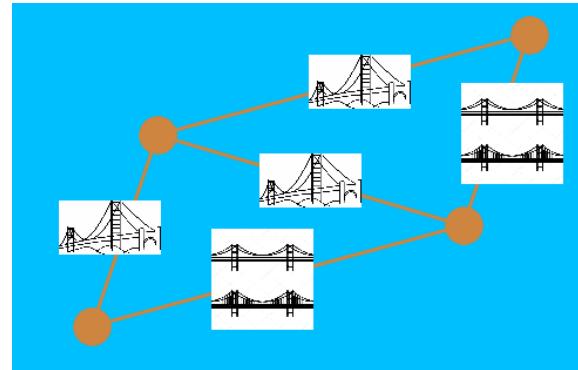
```

1 pos=nx.spectral_layout(G)
2 nx.draw_networkx_nodes(G, pos, node_color='grey', node_size=600)
3 nx.draw_networkx_edges(G, pos, edgelist=[(1,2), (2,3)], width=2, edge_color='deepskyblue')
4 nx.draw_networkx_edges(G, pos, edgelist=[(3, 4)], width=8, edge_color='lime')
5 nx.draw_networkx_labels(G, pos, font_size=16, font_color='white')
6
7 pos=nx.shell_layout(G)
8 nx.draw_networkx_nodes(G, pos, node_color='grey', node_size=600)
9 nx.draw_networkx_edges(G, pos, edgelist=[(1,2), (2,3)], width=2, edge_color='deepskyblue')
10 nx.draw_networkx_edges(G, pos, edgelist=[(3, 4)], width=8, edge_color='lime')
11 nx.draw_networkx_labels(G, pos, font_size=16, font_color='white')
12
13 pos=nx.fruchterman_reingold_layout(G)
14 nx.draw_networkx_nodes(G, pos, node_color='grey', node_size=600)
15 nx.draw_networkx_edges(G, pos, edgelist=[(1,2), (2,3)], width=2, edge_color='deepskyblue')
16 nx.draw_networkx_edges(G, pos, edgelist=[(3, 4)], width=8, edge_color='lime')
17 nx.draw_networkx_labels(G, pos, font_size=16, font_color='white')
```

## Multigrafo no dirigido cíclico



(a) Versión primaria



(b) spring\_layout

Figura 8: Ejemplo de Puentes de Königsberg

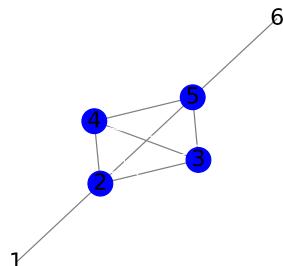
En la figura 8 se puede apreciar que ambos algoritmos dibujan visualmente bien el grafo, sin embargo esto es por que el ejemplo es pequeño. A continuación se muestra parte del código en Python para dibujar grafos que tengan imágenes en las aristas, basado en un código ejemplo [7].

```

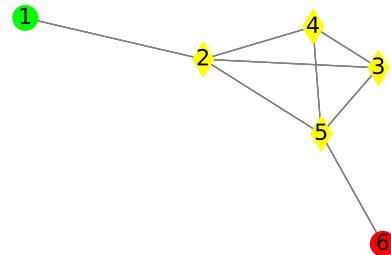
1 import matplotlib.pyplot as plt
2 import matplotlib.image as mpimg
3 pos=nx.spring_layout(G)
4 ax=plt.gca()
5 fig=plt.gcf()
6 label_pos = 0.5
7 trans = ax.transData.transform
8 trans2 = fig.transFigure.inverted().transform
9 imsize = 0.1
10 for (n1,n2) in G.edges():
11     (x1,y1) = pos[n1]
12     (x2,y2) = pos[n2]
13     (x,y) = (x1 * label_pos + x2 * (1.0 - label_pos),
14               y1 * label_pos + y2 * (1.0 - label_pos))
15     xx,yy = trans((x,y))
16     xa,ya = trans2((xx,yy))
17     imsize = G[n1][n2]['size']
18     img = G[n1][n2]['image']
19     a = plt.axes([xa-imsize/2.0,ya-imsize/2.0, imsize , imsize ])
20     a.imshow(img)

```

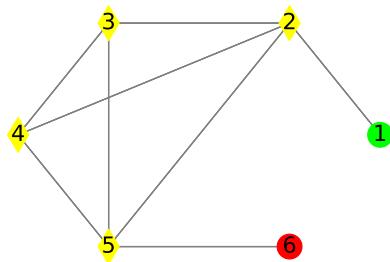
## Multigrafo no dirigido reflexivo



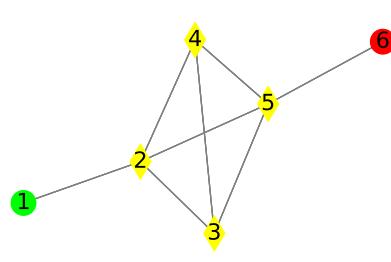
(a) Versión primaria



(b) `spring_layout`



(c) `shell_layout`

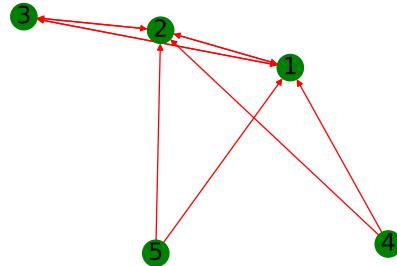


(d) `kamada_kawai_layout`

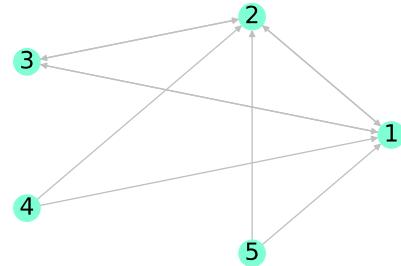
Figura 9: Ejemplo de proceso de calidad

En la figura 9 se puede apreciar que el algoritmo de acomodo que mejor dibuja la red es el algoritmo que da por default `NetworkX`, ya que por ejemplo, el algoritmo `shell_layout` no genera un dibujo del grafo bueno visualmente.

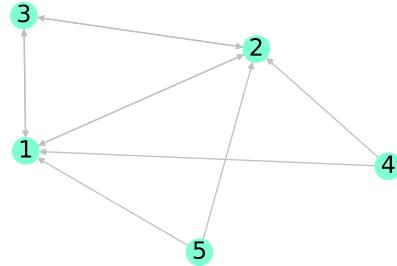
## Multigrafo dirigido acíclico



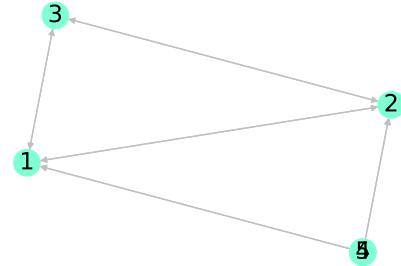
(a) Versión primaria



(b) `circular_layout`



(c) `fruchterman_reingold_layout`



(d) `kamada_kawai_layout`

Figura 10: Ejemplo de vuelos de avión

En la figura 10 se puede apreciar que el algoritmo de acomodo que mejor dibuja la red es el algoritmo `fruchterman_reingold_layout`, ya que por ejemplo, el algoritmo que da por default `NetworkX` y el algoritmo `kamada_kawai_layout`, no generan un dibujo del grafo bueno visualmente.



## Multigrafo dirigido cíclico

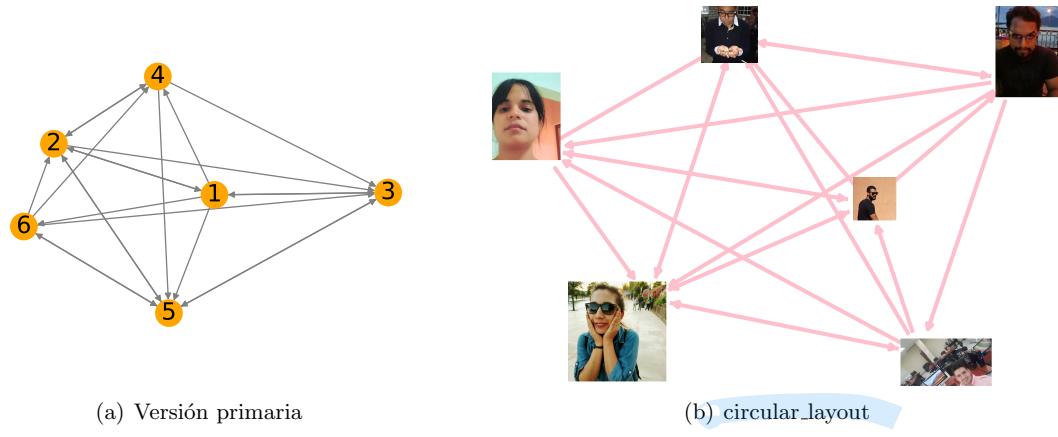


Figura 11: Ejemplo de redes sociales.

En la figura 10 se puede apreciar que el algoritmo de acomodo que mejor dibuja la red es el algoritmo `circular_layout`, ya que por ejemplo, el algoritmo que da por default `NetworkX` no genera un dibujo del grafo bueno visualmente.

## Multigrafo dirigido reflexivo

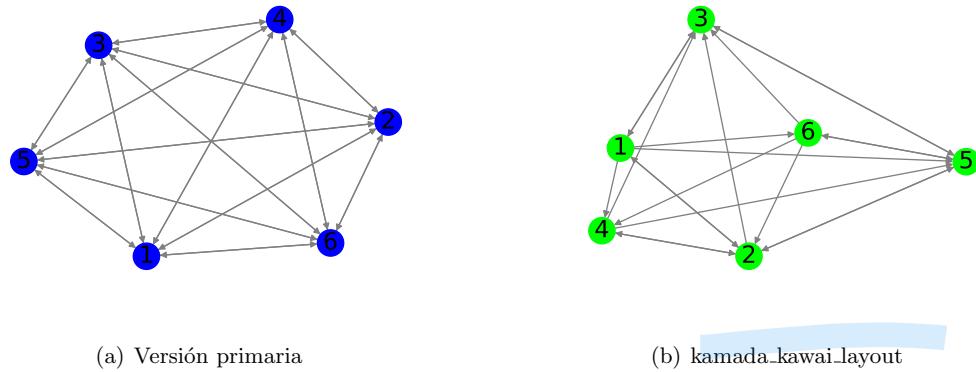


Figura 12: Ejemplo de transmisión de enfermedades

En la figura 10 se puede apreciar que el algoritmo de acomodo que mejor dibuja la red es el algoritmo que da por default `NetworkX`, ya que por ejemplo, el algoritmo `kamada_kawai_layout`, no generan un dibujo del grafo bueno visualmente.

## Referencias

- [1] R. Ahuja, T. Magnanti, and J. Orlin. Network Flows: Theory, Algorithms and Applications. [Prentice Hall], 1993.
- [2] M. Bazaraa, J. Jarvis, and H. Sherali. *Linear Programming and Network Flows*. Wiley, 4th edition edition, 2010.
- [3] Python. <https://www.python.org/>.
- [4] E. Schaeffer. <https://elisa.dyndns-web.com/teaching/opt/flow/>.
- [5] A. Serna. <https://github.com/sernarmando>.
- [6] Thomas MJ Fruchterman and Edward M Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.
- [7] <https://gist.github.com/shobhit/3236373>.
- [8] <https://www.geeksforgeeks.org/working-images-python/>.

T.M.J.

E.M.

# Medición de tiempo de ejecución

Matrícula: 1985281

9.5

## Resumen

Este trabajo busca resolver grafos con la implementación de los algoritmos de [NetworkX](#) de [Python](#). Se seleccionan algunos grafos de la tarea 2 y se generan otros, para así poder resolver cinco grafos distintos para cada uno de los cinco algoritmos seleccionados.

## Introducción

La librería [NetworkX](#) de [Python](#) proporciona algoritmos para resolver diversos problemas, de los cuales se escogieron los siguientes cinco:

- Centralidad intermedia: Calcula la centralidad de intermediación de rutas más cortas para todos los nodos.
- Árbol de expansión mínimo: Calcula un árbol de expansión mínimo del grafo.
- Flujo máximo: Calcula el flujo máximo de un nodo a otro.
- Ruta mas corta: Calcula las rutas más cortas entre los nodos del grafo.
- Coloración glotona: Colorea una gráfica usando varias estrategias de coloración codiciosa.



## Metodología y Resultados

Se seleccionan algunos grafos utilizados en la tarea 2 y se generan otros grafos que son compatibles con los requerimientos que el algoritmo tiene sobre sus datos de entrada, se repite cada ejecución por la cantidad suficiente de veces para que el tiempo total de ejecución del conjunto de réplicas sea mayor a cinco segundos (se sube el número de réplicas hasta que esto se logre). Luego se repite la medición del conjunto de réplicas hasta treinta veces en total, sin exceder quince minutos de tiempo total de ejecución.

Después con la ayuda de librerías adicionales tales como [NumPy](#) y [SciPy](#), se calcula para cada algoritmo el promedio ( $\mu$ ) y desviación estándar ( $\sigma$ ) de la ejecución del conjunto de réplicas y se grafica un histograma de las mediciones individuales para cada algoritmo.

```
1 import networkx as nx
2 import matplotlib.pyplot as plt
3 import time
4 import numpy as np
5 from scipy import stats
```

## Centralidad intermedia

La figura 1.b) muestra la centralidad intermedia para cada nodo del grafo ejemplo de redes sociales de la tarea 2.

```
1 b_w = nx.betweenness_centrality(G1)
```

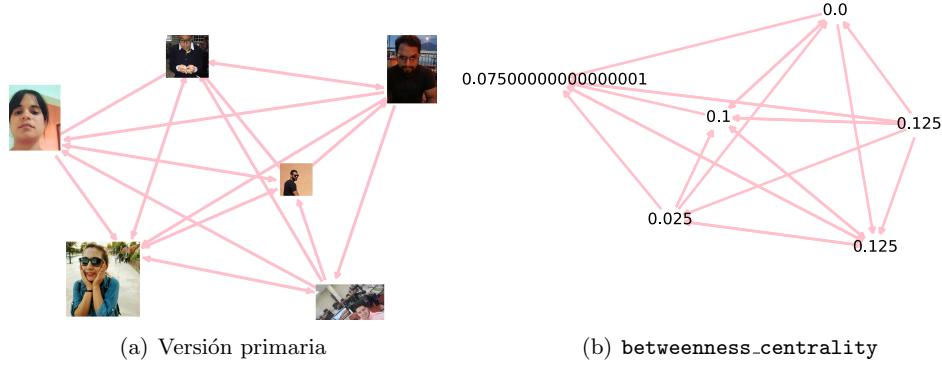


Figura 1: Ejemplo de redes sociales

También, otro grafo utilizado para aplicar este algoritmo es el famoso grafo de la red social *Krackhardt Kite*, una red social de diez actores presentada por David Krackhardt para ilustrar centralidad intermedia. Véase figura [2].

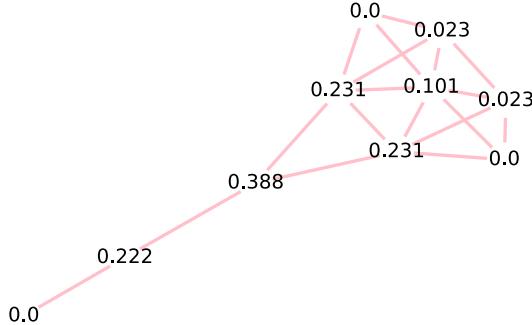


Figura 2: Red social de Krackhardt Kite

Ahora se calcula para el algoritmo, el promedio y desviación estándar de la ejecución del conjunto de réplicas y se grafica un histograma.

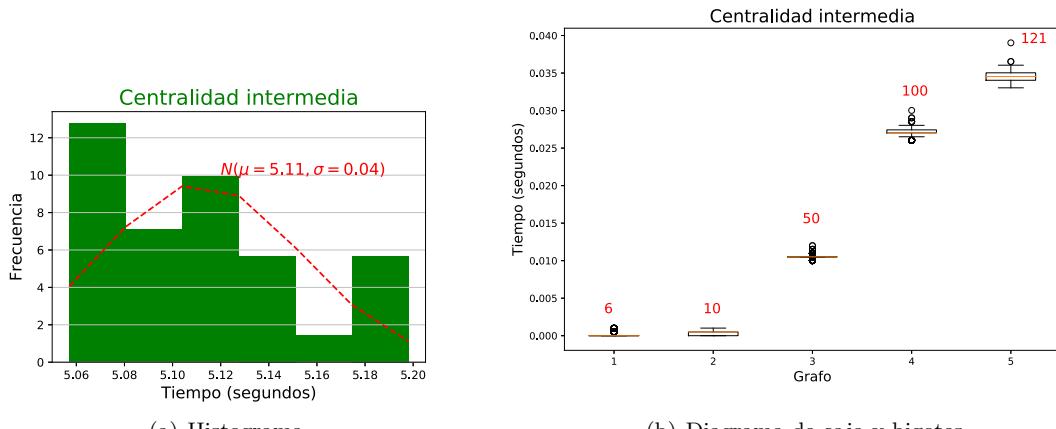


Figura 3: Histograma y diagrama de caja y bigotes

En la figura 3.a se muestra el histograma de los tiempos promedio del algoritmo y se calcula el promedio ( $\mu$ ) y la desviación estándar ( $\sigma$ ) para graficar la línea punteada que corresponde a la distribución normal  $N \sim (\mu = \frac{\sum_{i=1}^n x_i}{n} = 5.11, \sigma = \sqrt{\frac{\sum_{i=1}^n (x_i - \mu)^2}{n}} = 0.04)$  y se puede ver que la línea punteada no se ajusta al histograma por lo que se realiza la prueba estadística *Shapiro-Wilk* para checar la normalidad de los datos.

```
1 stats.shapiro(tiempos_algoritmo_1)
```

In: stats.shapiro(tiempos_algoritmo_1)	Out: ( $W = 0.6412, p = 2.4210 \cdot 10^{-7}$ )
--	---

La prueba muestra que el valor es menor a 0.05 (nivel de significancia), entonces la hipótesis nula es rechazada (se concluye que los datos no vienen de una distribución normal).

Por otro lado en la figura 3.b se muestran los diagramas de caja y bigotes para los tiempos individuales de los cinco grafos utilizados en el algoritmo, el número rojo representa la cantidad de nodos del grafo y se puede apreciar que conforme aumenta la cantidad de nodos en el grafo, los tiempos computacionales también crecen.

## Árbol de expansión mínimo

La figura ~~X4~~ muestra ejemplos de grafos y sus respectivos grafos de árboles de expansión mínima.

```
T1 = nx.minimum_spanning_tree(G6)
```

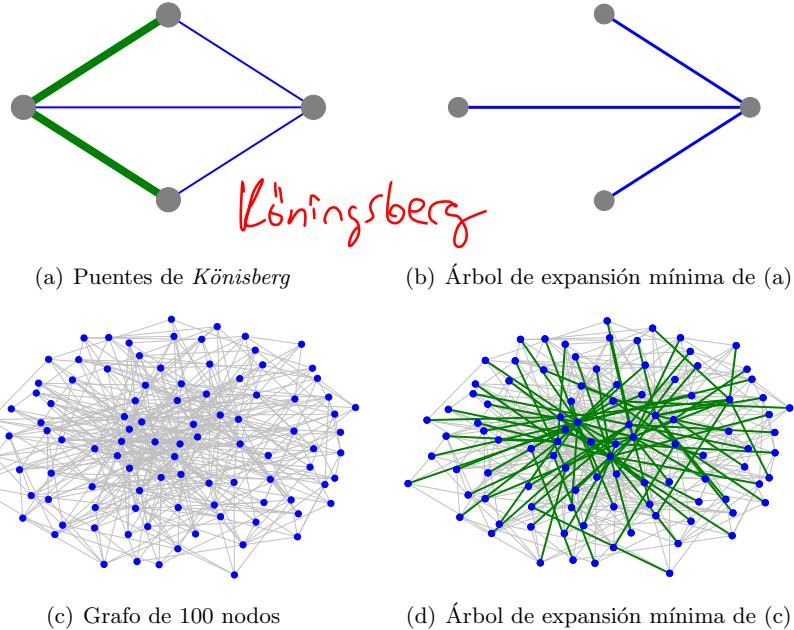


Figura 4: Ejemplos de árbol de expansión mínima

Ahora se calcula para el algoritmo, el promedio y desviación estándar de la ejecución del conjunto de réplicas y se grafica un histograma.

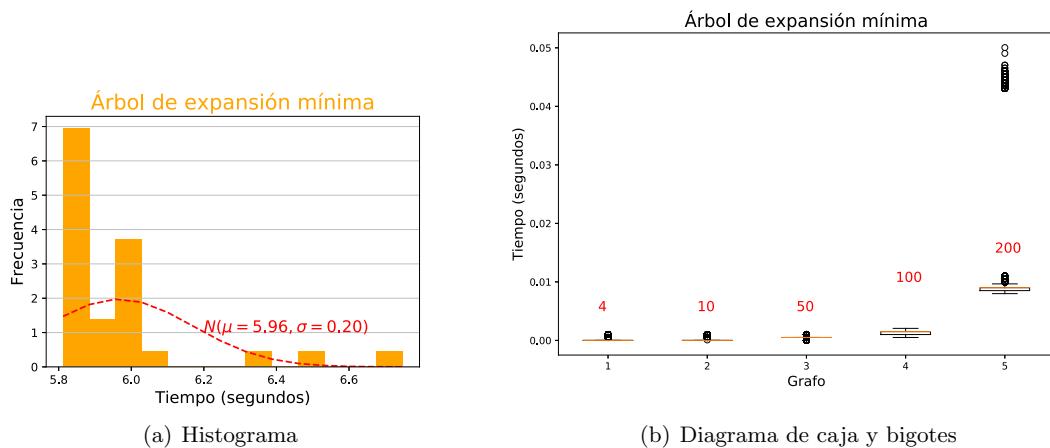


Figura 5: Histograma y diagrama de caja y bigotes

En la figura 5.a se muestra el histograma de los tiempos promedio del algoritmo y se calcula el promedio ( $\mu$ ) y la desviación estándar ( $\sigma$ ) para graficar la línea punteada que corresponde a la distribución normal  $N \sim (\mu = 5.96, \sigma = 0.20)$  y se puede ver que la línea punteada no se ajusta al histograma por lo que se realiza la prueba estadística *Shapiro-Wilk* para checar la normalidad de los datos.

In: stats.shapiro (tiempos_algoritmo_2)	Out: ( $W = 0.5320, p = 1.1895 \cdot 10^{-8}$ )
---	---

La prueba muestra que el valor es menor a 0.05 (nivel de significancia), entonces la hipótesis nula es rechazada (se concluye que los datos no vienen de una distribución normal).

Por otro lado en la figura 5.b se muestran los diagramas de caja y bigotes para los tiempos individuales de los cinco grafos utilizados en el algoritmo, el número rojo representa la cantidad de nodos del grafo y se puede apreciar que conforme aumenta la cantidad de nodos en el grafo, los tiempos computacionales también crecen.

### Flujo máximo

En este ejemplo se le pide al algoritmo de flujo máximo encontrar el flujo máximo del nodo uno al nodo diez del grafo llamado G11.

1 MF_1 = nx.maximum_flow(G11, 1, 10, capacity='capacidad', flow_func=None)
--

Ahora se calcula para el algoritmo, el promedio y desviación estándar de la ejecución del conjunto de réplicas y se grafica un histograma.

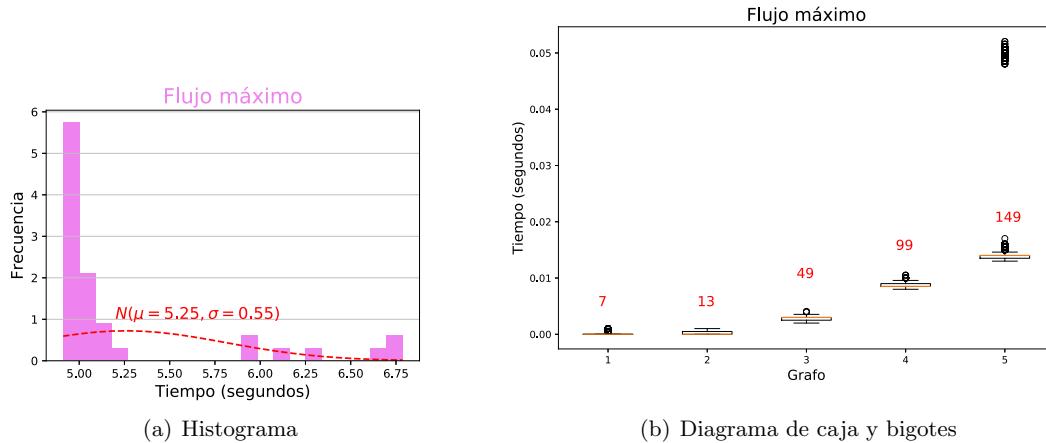


Figura 6: Histograma y diagrama de caja y bigotes

En la figura 6.a se muestra el histograma de los tiempos promedio del algoritmo y se calcula el promedio ( $\mu$ ) y la desviación estándar ( $\sigma$ ) para graficar la línea punteada que corresponde a la distribución normal  $N \sim (\mu = 5.25, \sigma = 0.55)$  y se puede ver que la línea punteada no se ajusta al histograma por lo que se realiza la prueba estadística *Shapiro-Wilk* para checar la normalidad de los datos.

In: stats.shapiro (tiempos\_algoritmo\_3)

Out: ( $W = 0.8749, p = 0.0021$ )

La prueba muestra que el p-valor es menor a 0.05 (nivel de significancia), entonces la hipótesis nula es rechazada (se concluye que los datos no vienen de una distribución normal).

Por otro lado en la figura [6.b] se muestran los diagramas de caja y bigotes para los tiempos individuales de los cinco grafos utilizados en el algoritmo, el número rojo representa la cantidad de nodos del grafo y se puede apreciar que conforme aumenta la cantidad de nodos en el grafo, los tiempos computacionales también crecen.

### Ruta mas corta

La figura [7] muestra un ejemplo de un grafo y sus respectivas rutas más cortas del nodo uno al nodo siete del grafo llamado G16.

```
1 rutas1 = [p for p in nx.all_shortest_paths(G16, source=1, target=7, weight=None, method='dijkstra')]
```

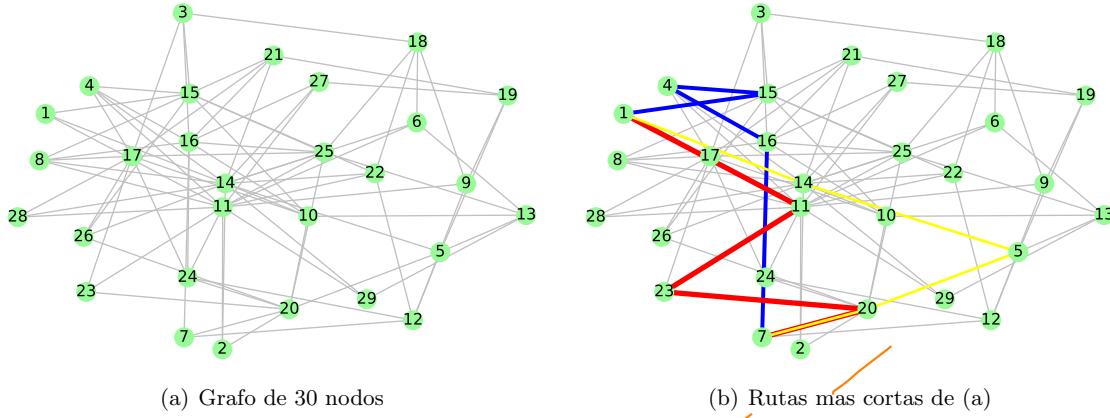


Figura 7: Ejemplo de rutas más cortas

Ahora se calcula para el algoritmo, el promedio y desviación estándar de la ejecución del conjunto de réplicas y se grafica un histograma.

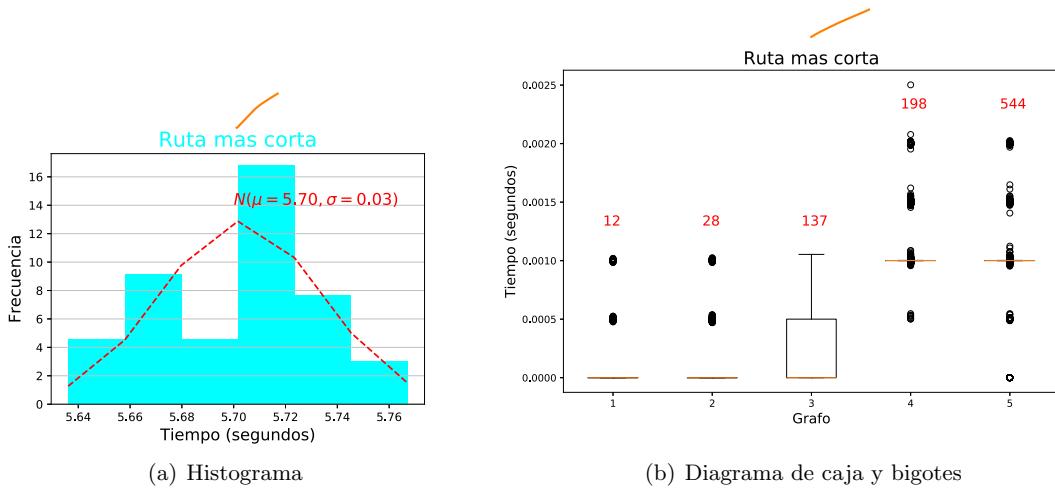


Figura 8: Histograma y diagrama de caja y bigotes

En la figura 8.a se muestra el histograma de los tiempos promedio del algoritmo y se calcula el promedio ( $\mu$ ) y la desviación estándar ( $\sigma$ ) para graficar la línea punteada que corresponde a la distribución normal  $N \sim (\mu = 5.70, \sigma = 0.03)$  y se puede ver que la línea punteada no se ajusta al histograma por lo que se realiza la prueba estadística *Shapiro-Wilk* para checar la normalidad de los datos.

<b>In:</b> stats.shapiro(tiempos_algoritmo_4)	<b>Out:</b> ( $W = 0.7345, p = 5.1768 \cdot 10^{-6}$ )
---	--

La prueba muestra que el  $p$ -valor es menor a 0.05 (nivel de significancia), entonces la hipótesis nula es rechazada (se concluye que los datos no vienen de una distribución normal).

Por otro lado en la figura 8.b se muestran los diagramas de caja y bigotes para los tiempos individuales de los cinco grafos utilizados en el algoritmo, el número rojo representa la cantidad de nodos del grafo y se puede apreciar que conforme aumenta la cantidad de nodos en el grafo, los tiempos computacionales también crecen.

### Coloración glotona

La figura 9 muestra un ejemplo de un grafo y sus respectiva coloración propuesta por el algoritmo ejemplo.

1	greedy1 = nx.coloring.greedy_color(G21, strategy='random_sequential')
---	---

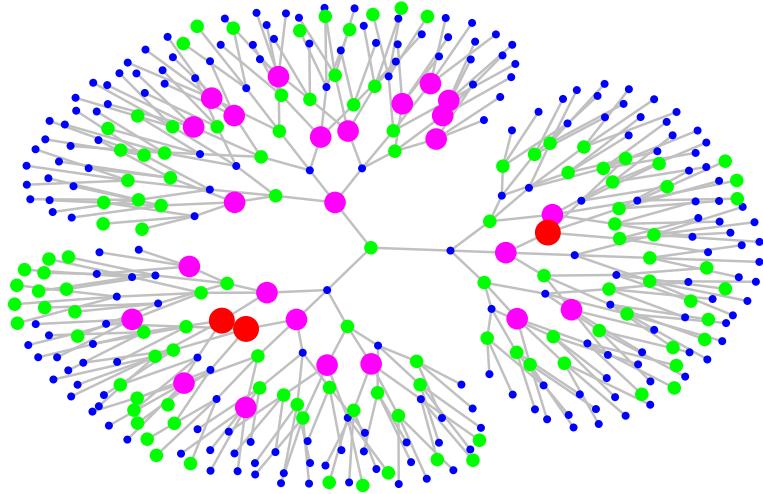


Figura 9: Ejemplo de coloración random

Ahora se calcula para el algoritmo, el promedio y desviación estándar de la ejecución del conjunto de réplicas y se grafica un histograma.

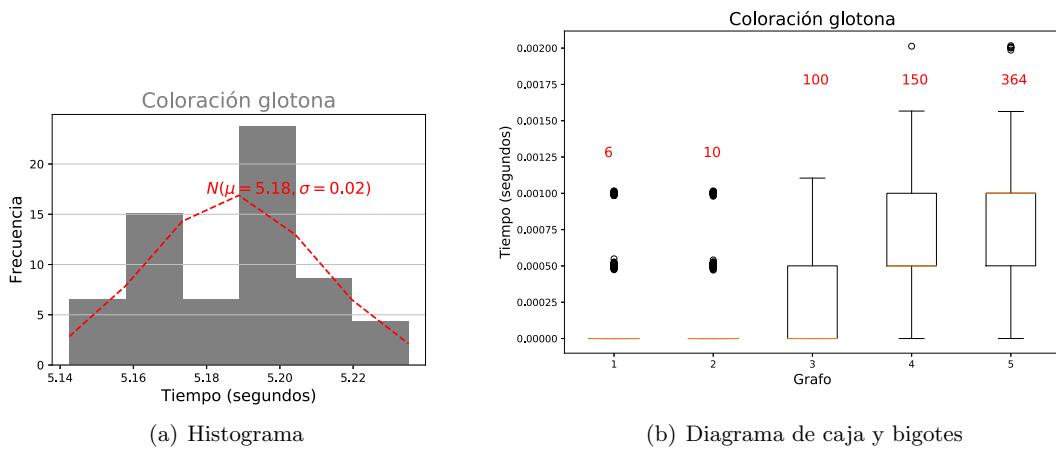


Figura 10: Histograma y diagrama de caja y bigotes

En la figura 10.a) se muestra el histograma de los tiempos promedio del algoritmo y se calcula el promedio ( $\mu$ ) y la desviación estándar ( $\sigma$ ) para graficar la línea punteada que corresponde a la distribución normal  $N \sim (\mu = 5.18, \sigma = 0.02)$  y se puede ver que la línea punteada no se ajusta al histograma por lo que se realiza la prueba estadística *Shapiro-Wilk* para checar la normalidad de los datos.

In: stats.shapiro (tiempos_algoritmo_5)	Out: ( $W = 0.5863, p = 4.9988 \cdot 10^{-8}$ )
---	---

La prueba muestra que el p-valor es menor a 0.05 (nivel de significancia), entonces la hipótesis nula es rechazada (se concluye que los datos no vienen de una distribución normal).

Por otro lado en la figura [10.b] se muestran los diagramas de caja y bigotes para los tiempos individuales de los cinco grafos utilizados en el algoritmo, el número rojo representa la cantidad de nodos del grafo y se puede apreciar que conforme aumenta la cantidad de nodos en el grafo, los tiempos computacionales también crecen.

Además se incluyen dos gráficas de dispersión, en ambas el eje horizontal corresponde al tiempo promedio de ejecución de un algoritmo, contando con barras horizontales que representan la desviación estándar alrededor del punto que indica el promedio, mientras el eje vertical es el número de vértices del grafo en la primera gráfica, ver figura [11] y el número de aristas en la segunda, ver figura [12]; cada combinación de algoritmo-grafo se visualiza con un punto en la gráfica de dispersión (se distinguen entre algoritmos por colores y entre grafos por formas).

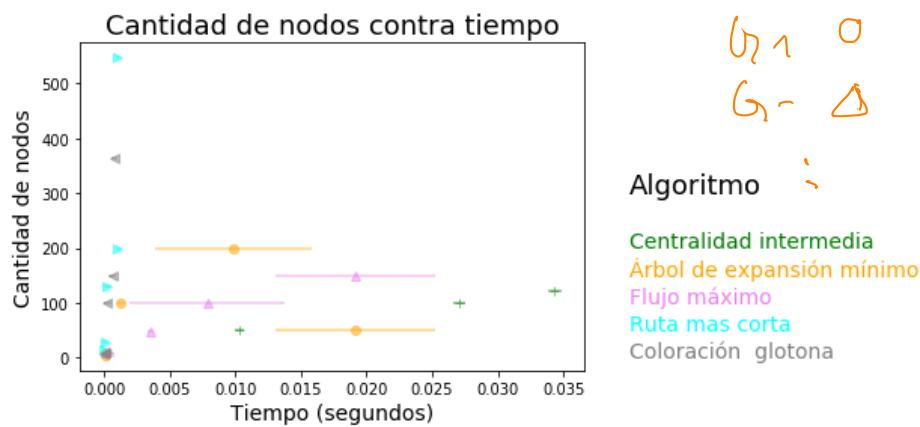


Figura 11: Cantidad de nodos contra tiempos

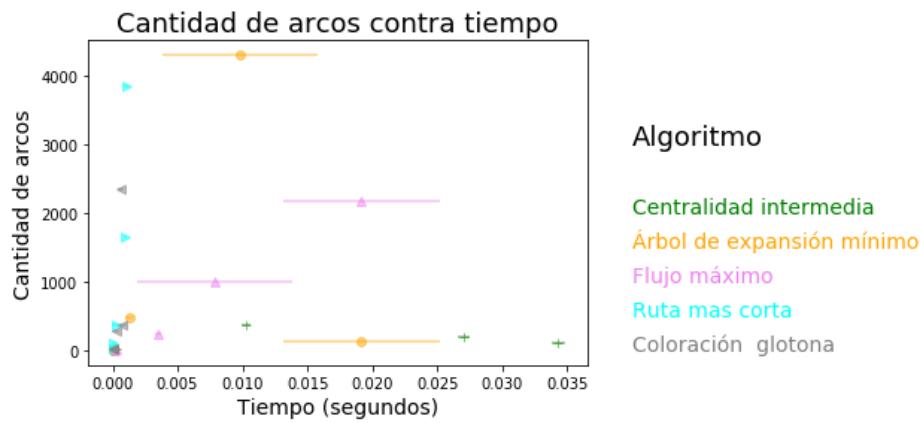


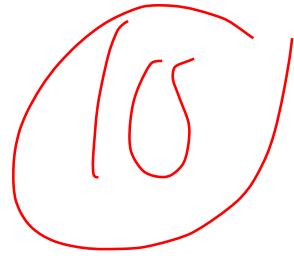
Figura 12: Cantidad de arcos contra tiempos

## Referencias

- [1] Python. <https://www.python.org/>.
- [2] E. Schaeffer. <https://elisa.dyndns-web.com/teaching/opt/flow/>.
- [3] A. Serna. <https://github.com/sernarmando/>

# Complejidad asintótica experimental

Matrícula: 1985281



## Resumen

Este trabajo busca resolver grafos con la implementación de los algoritmos de [NetworkX](#) de [Python](#). Primero se seleccionan tres métodos de generación de grafos, luego con cada generador, se generan grafos de cuatro diferentes órdenes en escala logarítmica (16, 32, 64 y 128 en base 2), después se generan diez grafos distintos de cada orden.

## Introducción

La librería [NetworkX](#) de [Python](#) proporciona algoritmos y generadores de grafos, si los grafos generados son ponderados o multigrafos, se convierten a grafos simples. Además se asignan pesos no-negativos normalmente distribuidos (con media  $\mu = 1$  y desviación  $\sigma = 0$ ) a las aristas para que se puedan utilizar como instancias del problema de flujo máximo.

Luego se eligen tres implementaciones de [NetworkX](#) de los algoritmos de flujo máximo, se ejecutan los algoritmos seleccionados con cinco diferentes pares de fuente-sumidero por lo menos cinco veces cada par  $s - t$ , cada grafo de cada tamaño con cada algoritmo.

### Generadores:

- `dense_gnm_random_graph` (Generador 1): Devuelve un  $G_{n,m}$  grafo aleatorio, es decir, se elige de manera uniformemente al azar del conjunto de todos los grafos con  $n$  nodos y  $m$  arcos.
- `gnp_random_graph` (Generador 2): Devuelve un  $G_{n,p}$  grafo aleatorio, también conocido como un grafo de [Erdős-Rényi](#) o un grafo binomial, es decir, se elige cada uno de los arcos posibles con probabilidad  $p$ .
- `gnm_random_graph` (Generador 3): Devuelve un  $G_{n,m}$  grafo aleatorio.

### Algoritmos:

- `maximum_flow` (Algoritmo 1): Encuentra el flujo máximo de un nodo fuente a un nodo sumidero.
- `algorithms.flow.edmonds_karp` (Algoritmo 2): Encuentra el flujo máximo de un nodo fuente a un nodo sumidero utilizando el algoritmo [Edmonds-Karp](#). Este algoritmo tiene un tiempo de ejecución de  $O(nm^2)$  para  $n$  nodos y  $m$  arcos.
- `algorithms.flow.boycov_kolmogorov` (Algoritmo 3): Encuentra el flujo máximo de un nodo fuente a un nodo sumidero utilizando el algoritmo [Bojkov-Kolmogorov](#). Este algoritmo tiene complejidad  $O(n^2m|C|)$  para  $n$  nodos  $m$  bordes, y  $|C|$  es el coste del corte mínimo.

## Metodología y Resultados

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

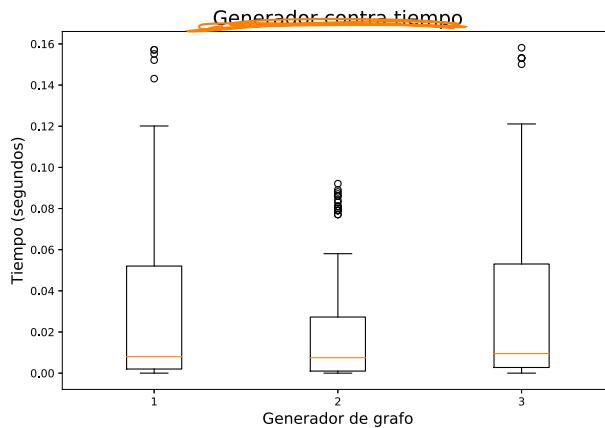


Figura 1: Tipo de generador de grafo contra tiempo

En la figura 1 se puede apreciar que el generador más rápido de los tres seleccionados es el Generador 2, mientras que los otros dos parecen ser igual de tardados.

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

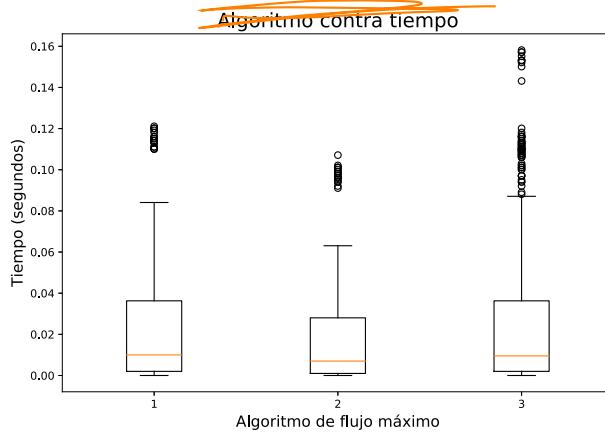


Figura 2: Algoritmo de flujo máximo contra tiempo

En la figura 2 se puede apreciar que el algoritmo más rápido de los tres seleccionados es el Algoritmo 2, mientras que el Algoritmo 3 parece ser el más tardado de los tres.

## Efecto que el orden del grafo tiene en el tiempo de ejecución

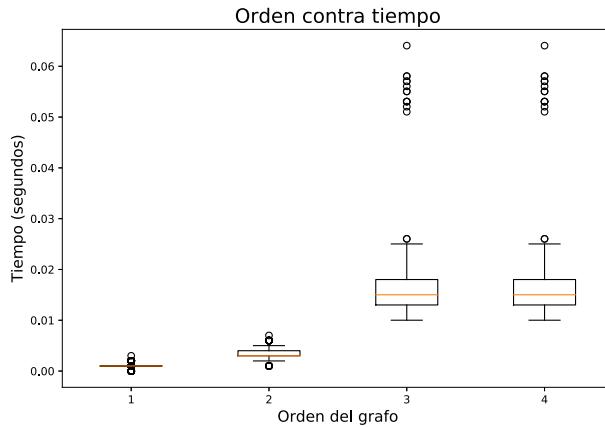


Figura 3: Orden del grafo contra tiempo

En la figura 3 se puede apreciar que conforme el orden del grafo va creciendo ( $2^4, 2^5, 2^6, 2^7$ ) respectivamente, el tiempo de ejecución también va creciendo.

## Efecto que la densidad del grafo (~~como tasa de aristas presentes entre aristas posibles~~) tiene en el tiempo de ejecución

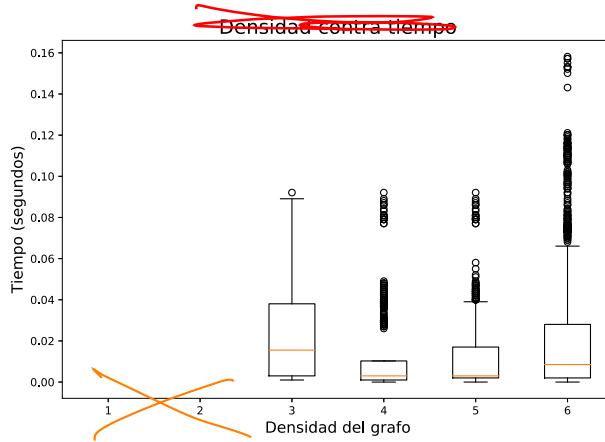


Figura 4: Densidad del grafo contra tiempo

En la figura 4 se puede apreciar que conforme la densidad del grafo va creciendo ( $[0, 0.5], (0.5, 0.6], (0.6, 0.7], (0.7, 0.8], (0.8, 0.9], (0.9, 1]$ ) respectivamente, el tiempo de ejecución va disminuyendo.

Se puede observar que los diagramas de caja y bigotes de las figuras 1, 2, 3 y 4, no muestran tan profundamente si sí o no estos efectos tienen interacciones, por lo que se sigue a realizar un análisis de varianza para cada una de las variables.

	sum_sq	df	F	PR>F	
Algoritmo	0.00163	1	4.66000	3.089750e-02	0.031
Generador	0.00229	1	9.77599	1.796692e-03	0.002
Orden	1.21838	1	5249.0746	0.000000e+00	0] < 0.001
Densidad	0.04643	1	199.05200	6.059348e-43	< 0.001
Residual	0.416630	1795	Nan	Nan	0.001
cobertura	0.111750	1795	Nan	Nan	

Tenemos que el p-valor obtenido por la prueba ANOVA es muy pequeño, es decir, que las medias de las variables con respecto al tiempo son distintas, entonces se puede decir que las variables están relacionadas con los tiempos de ejecución. Una vez hecha la ANOVA se debe verificar que todos los datos deben distribuirse normalmente para que la estadística  $F$  sea confiable.

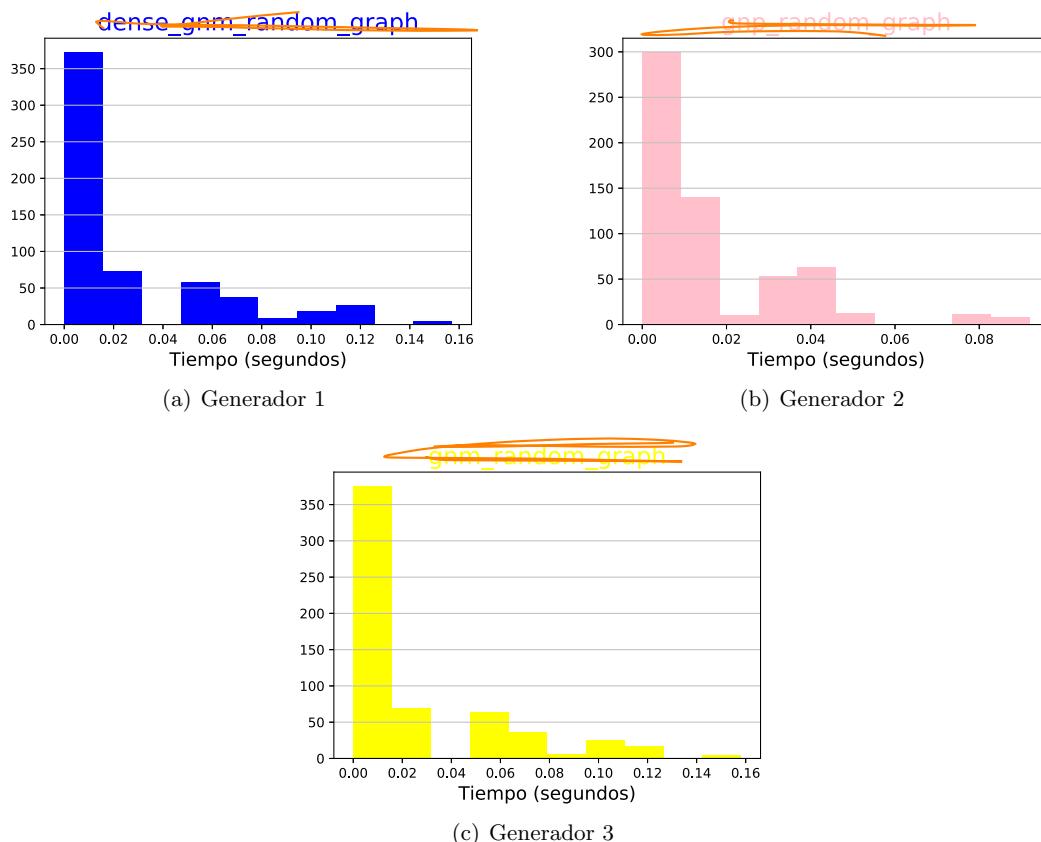


Figura 5: Histogramas de los tiempos por tipo de generador de grafo

De la figura 5 es fácil ver que los datos de los tiempos de los generados de grafos no son distribuidos normalmente.

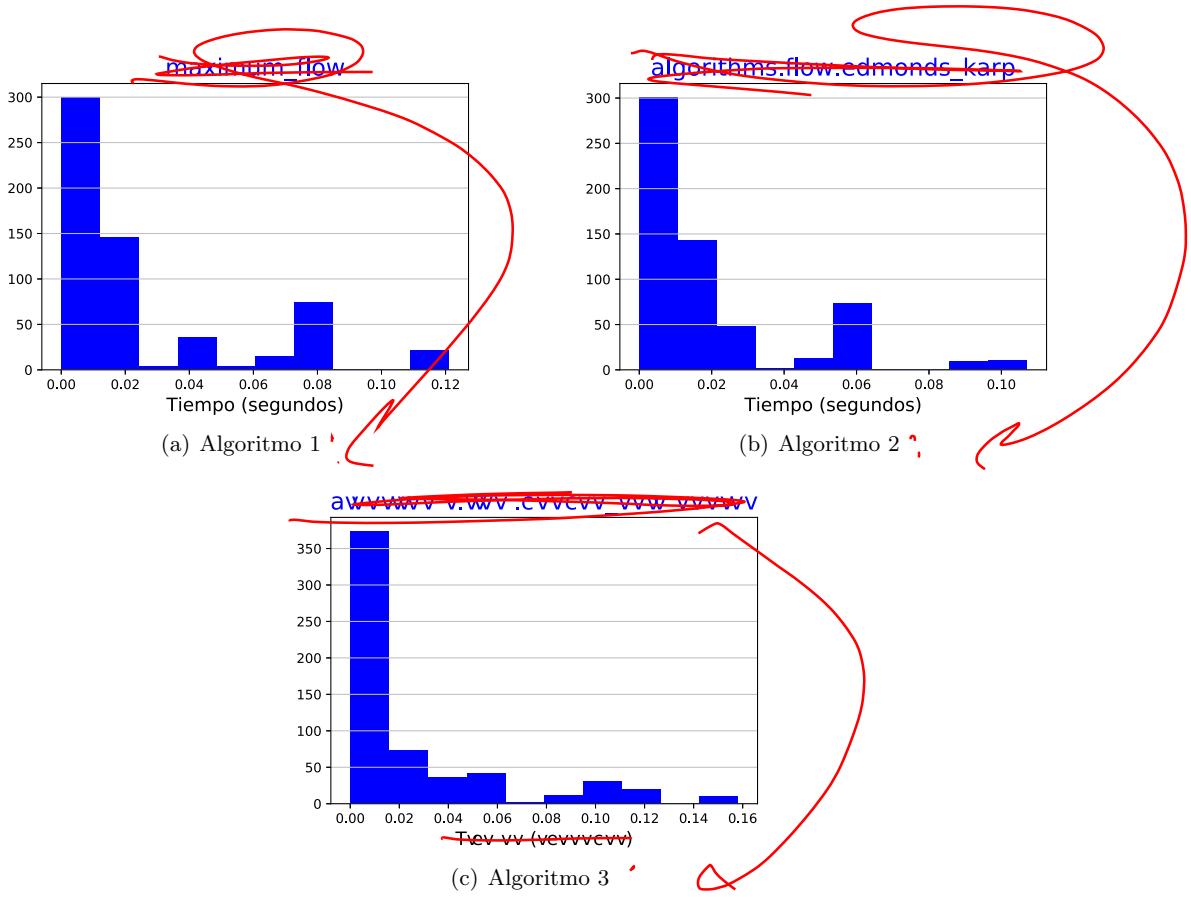
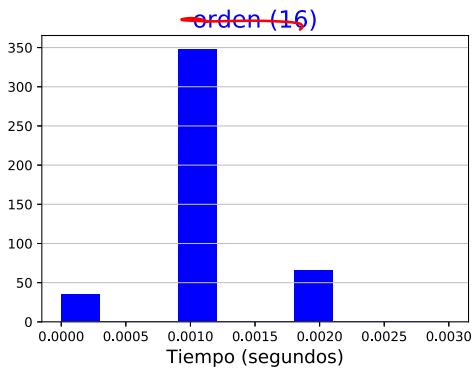
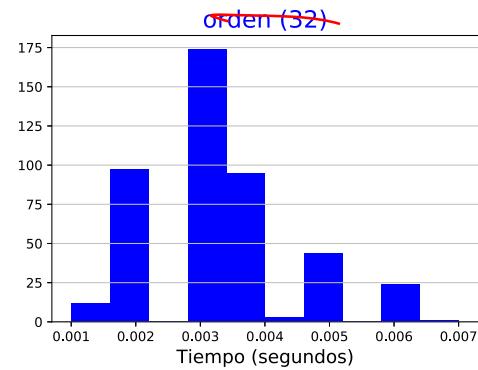


Figura 6: Histogramas de los tiempos por tipo de algoritmo de flujo máximo

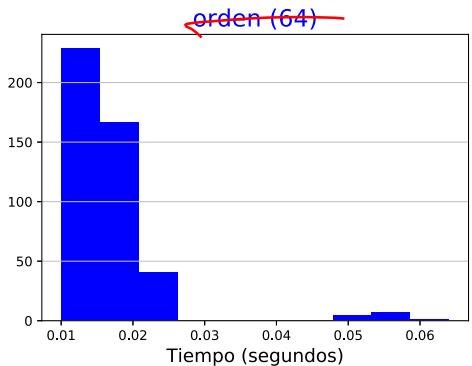
De la figura 6 es fácil ver que los datos de los tiempos de los algoritmos de flujo máximo no son distribuidos normalmente.



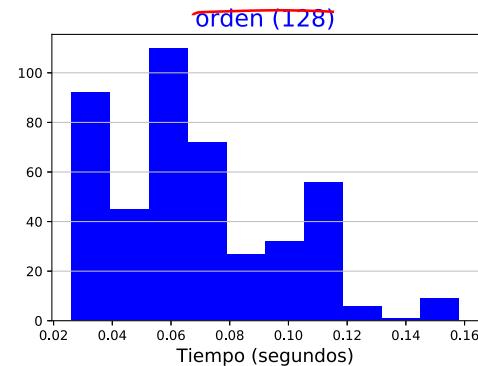
(a) Orden (16)



(b) Orden (32)



(c) Orden (64)



(d) Orden (128)

Figura 7: Histogramas de los tiempos por tamaño de orden del grafo

De la figura 7 es fácil ver que los datos de los tiempos por tamaño del orden de los grafos no son distribuidos normalmente.

De las figuras 5, 6, y 7, podemos ver que los tiempos de ejecución no distribuyen normalmente para las variables por lo que se propone para estudiar más a fondo la relación entre las variables una prueba de Mínimos Cuadrados (OLS).

## OLS Regression Results

Dep. Variable:	Tiempo	R-squared:	0.758
Model:	OLS	Adj. R-squared:	0.758
Method:	Least Squares	F-statistic:	1408.0
Date:	Mon, 01 Apr 2019	Prob F-statistic:	0.00
Time:	22:50:42	Log-Likelihood:	4979.9
No. Observations:	1800	AIC:	-9950.0
Df Residuals:	1795	BIC:	-9922.0
Df Model:	4		
Covariance Type:	nonrobust		
coef	std err	t	P> t
Intercept	-0.0421	0.002	-18.436
Algoritmo	0.0010	0.000	2.160
Generador	-0.0014	0.000	-3.127
Orden	0.0006	8.39e-06	72.450
Densidad	0.0329	0.002	14.109
Omnibus:	705.514	Durbin-Watson:	1.147
ProbOmnibus:	0.000	Jarque-Bera JB:	3663.472
Skew:	1.785	ProbJB:	0.00
Kurtosis:	9.009	Cond. No.	635.

## Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.  
 c@FancyVerbLineors assume that the covariance matrix of the errors is correctly specified.

*n^2*

La prueba de Mínimos Cuadros muestra con la *Muñeca* que se puede eliminar hasta el 75 % de errores para determinar el tiempo, por otro lado, el *p – valor* muestra que las medias de las variables son distintas respecto al tiempo y por último podemos ver que la variable Orden está más relacionada con los tiempos de ejecución, luego le sigue el Densidad del grafo y después el Algoritmo de flujo máximo. Para rectificar la prueba de Mínimos Cuadrados se hace una Matriz de Correlaciones. Ver figura 8.

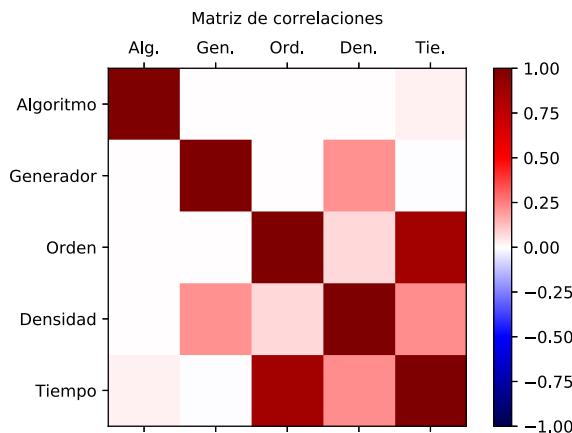


Figura 8: Matriz de correlaciones

Por último se representan en la figura 9 todos los grafos y se identifican según su generador por color y su algoritmo de solución por forma.

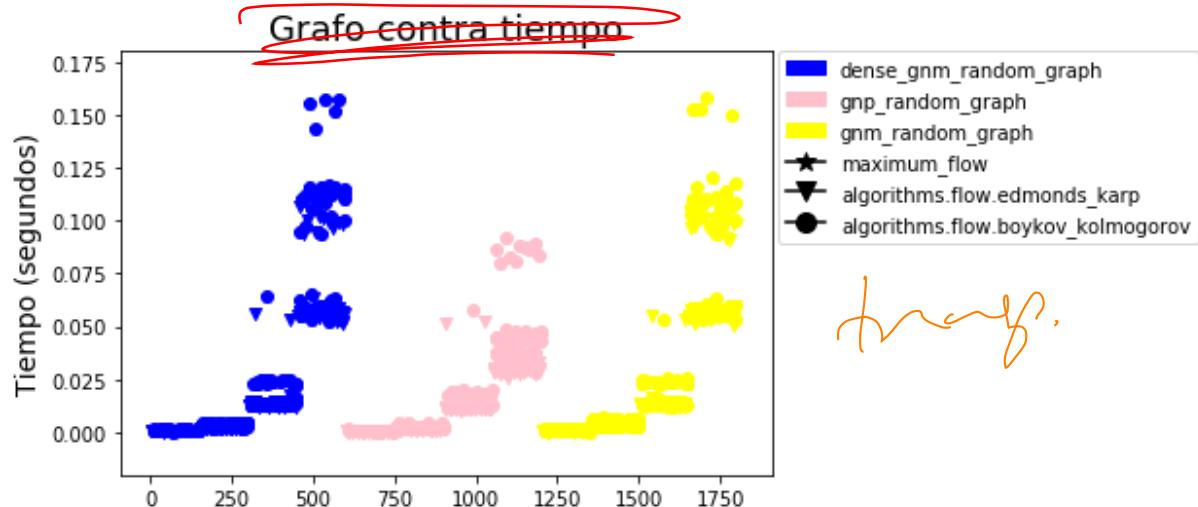


Figura 9: Grafos contra tiempo

## Referencias

- [1] Python. <https://www.python.org/>.
- [2] E. Schaeffer. <https://elisa.dyndns-web.com/teaching/opt/flow/>.
- [3] A. Serna. <https://github.com/sernarmando>.

title = ...

# Caracterización estructural de instancias

Matrícula: 1985281



## Resumen

Entre los generadores de grafos que ~~proporciona~~ NetworkX, se seleccionó `random_geometric_graph` (grafo geométrico aleatorio) y la función `preflow_push` para calcular el flujo máximo entre un par de nodos, ya que resulta ser la más eficiente para este tipo de grafos. Además, se escogieron los grafos geométricos aleatorios, ya que se suelen utilizar para representar redes de sensores inalámbricos (inglés: wireless sensor networks (WSN)).

## Introducción

Las redes de sensores inalámbricos se componen de nodos representados por los sensores, los cuales poseen una capacidad limitada de computación y comunicación. Por otro lado, la gran cantidad de sensores sobre una región dificulta la posibilidad de la colocación estratégica de nuevos dispositivos, y en consecuencia, la implementación aleatoria suele ser la mejor opción. Para este tipo de problemas se suele representar los sensores mediante puntos aleatorios finitos sobre la región, en donde las redes de sensores inalámbricos se modelan como grafos geométricos aleatorios que dependen del comportamiento probabilístico a estudiar.

## Metodología y Resultados

Se visualizan cinco de las instancias producidas por el generador seleccionado y se visualizan con un acomodo que depende de las características del grafo, cambiando el tamaño y color de los nodos *fuente* (color verde) y *sumidero* (color rojo) en las visualizaciones, igual como el grosor de los arcos que son linealmente proporcionales a sus capacidades, ver figura 1.

```
1 G1 = nx.random_geometric_graph(50, 0.25)
2 pos1 = nx.get_node_attributes(G1, 'pos')
3 weights1 = np.random.normal(3, 1, nx.number_of_edges(G1))
4 w = 0
5 for u, v, d in G1.edges(data=True):
6     d['weight'] = weights1[w]
7     w += 1
8 f1 = {randint(0,49)}
9 s1 = {randint(0,49)}
10 nx.draw(G1, node_color='blue', edge_color='silver', node_size=80, width=weights1,
11         pos=pos1, with_labels=False, alpha= 0.7)
12 nx.draw_networkx_nodes(G1, pos1, nodelist=f1, node_size=150, node_color='green',
13                       node_shape='d')
14 nx.draw_networkx_nodes(G1, pos1, nodelist=s1, node_size=150, node_color='red',
15                       node_shape='d')
```

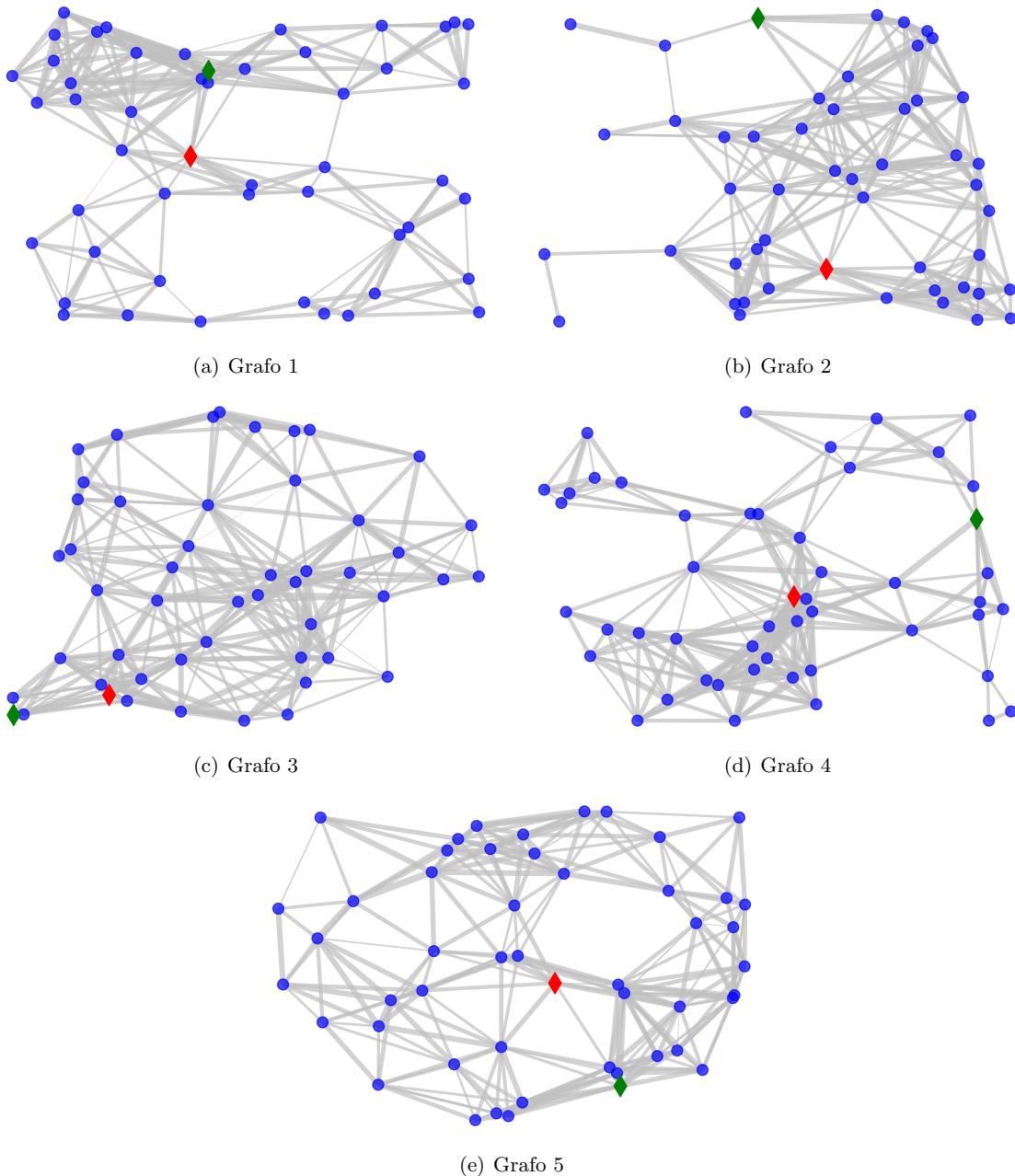


Figura 1: Ejemplo de cinco grafos geométricos aleatorios

Con los algoritmos disponibles en [NetworkX](#), se calcula para las cinco instancias las siguientes características estructurales para todos sus vértices:

1. **Distribución de grado:** Calcula la distribución de probabilidad en los grados de los nodos en el grafo.
2. **Coeficiente de agrupamiento:** Calcula el número de triadas posibles para los nodos en el

grafo.

3. **Centralidad de cercanía:** Calcula la centralidad de cercanía de un nodo  $u$  como el recíproco de la suma de las distancias del camino más corto desde todos los  $n - 1$  demás nodos.
4. **Centralidad de carga:** Calcula la centralidad de carga de un nodo  $u$  como la fracción de todas las rutas más cortas que pasan a través de ese nodo.
5. **Excentricidad:** Calcula la excentricidad de un nodo  $u$  como la distancia máxima de  $u$  a todos los demás nodos en el grafo.
6. **PageRank:** Calcula una clasificación de los nodos en el grafo en función de la estructura de los enlaces entrantes.

Cuadro 1: Instancias por características de sus nodos fuente y sumidero.

Grafo	Nodos		Característica estructural						Máximo Flujo
			1	2	3	4	5	6	
1	Fuente	49	5	0.40	0.28	0.024	6	0.022	2.48
	Sumidero	26	11	0.61	0.32	0.072	5	0.023	
2	Fuente	29	3	0.33	0.28	0.077	5	0.020	4.19
	Sumidero	12	5	0.90	0.29	0.003	6	0.018	
3	Fuente	22	8	0.67	0.37	0.050	5	0.018	5.31
	Sumidero	4	11	0.52	0.37	0.033	5	0.020	
4	Fuente	28	5	0.70	0.28	0.020	7	0.018	3.53
	Sumidero	2	5	0.60	0.35	0.079	5	0.015	
5	Fuente	5	3	0.70	0.28	0.026	6	0.018	3.06
	Sumidero	0	4	0.40	0.27	0.076	6	0.017	

## Distribución de grado

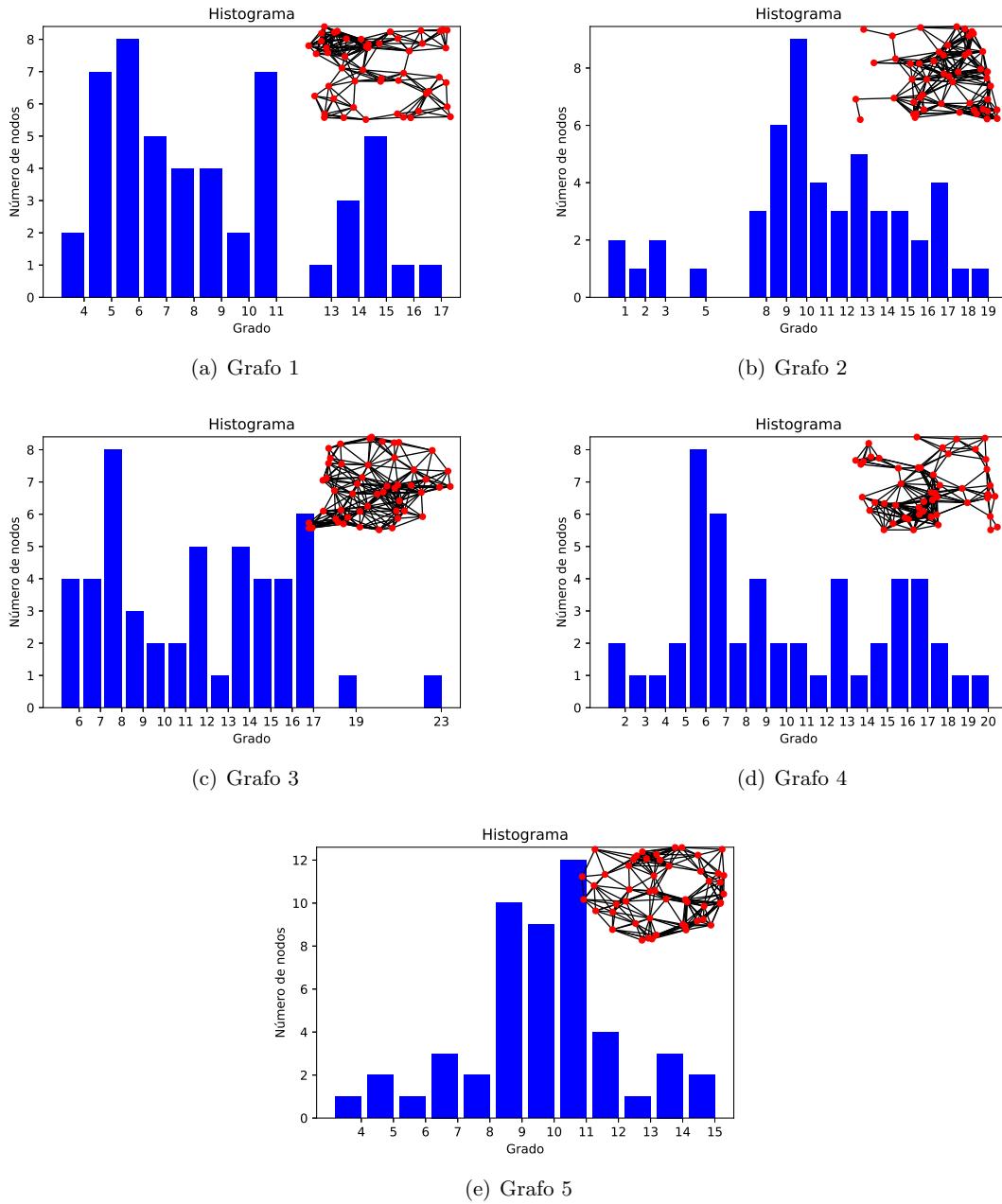


Figura 2: Histogramas de distribuciones de grado

## Coefficiente de agrupamiento

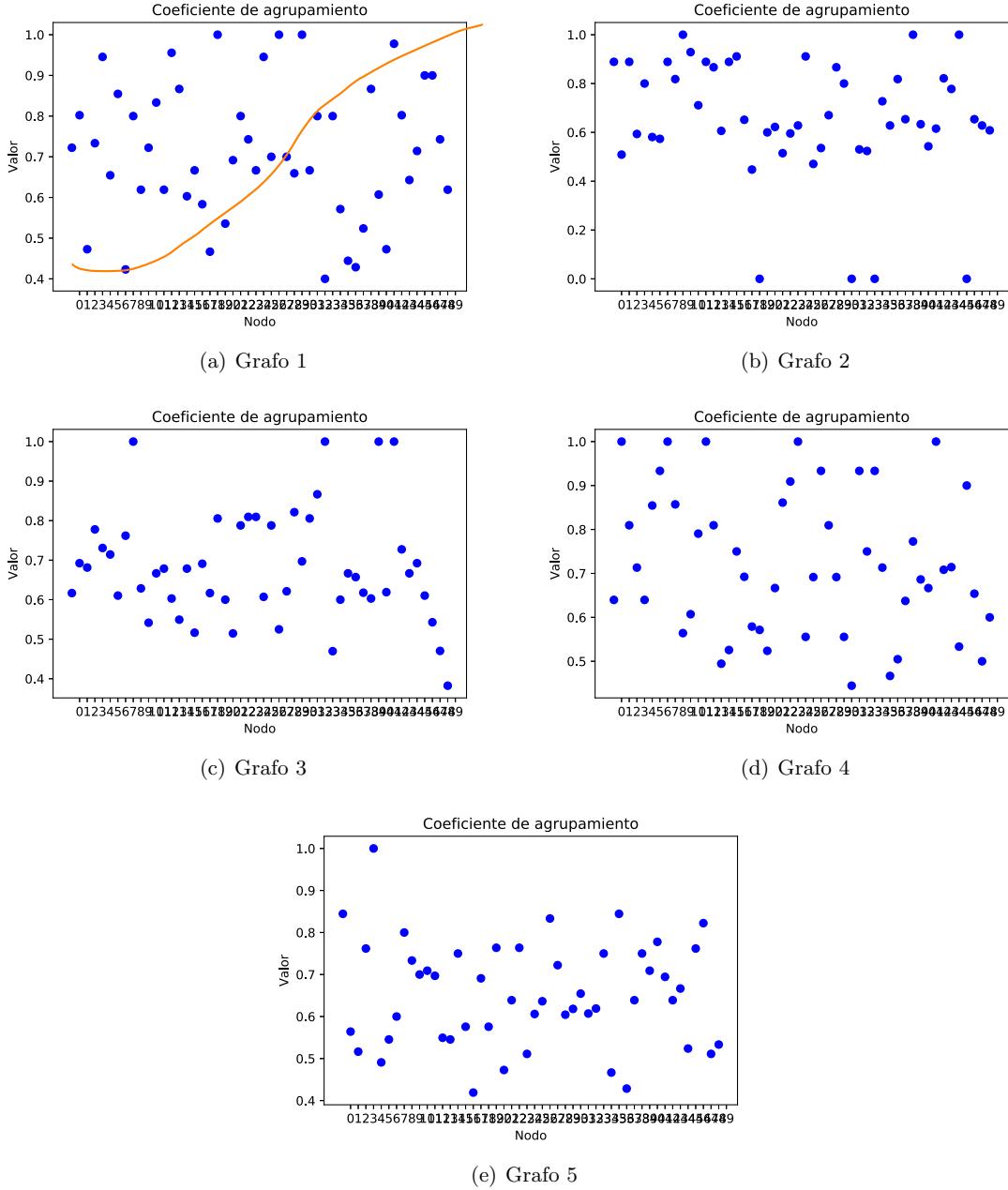
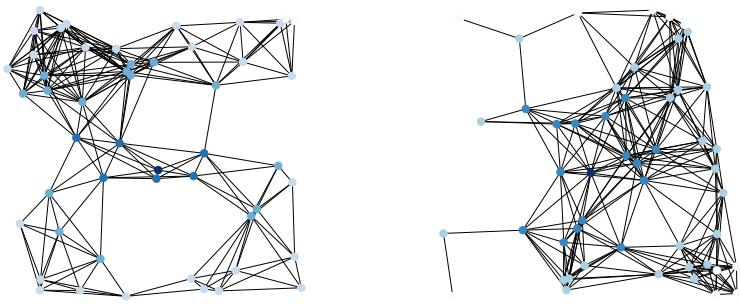


Figura 3: Coeficiente de agrupamiento

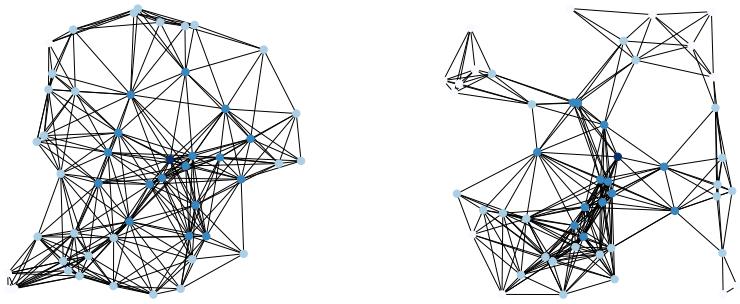
## Centralidad de cercanía

La figura 4 colorea los nodos dependiendo su valor de centralidad de cercanía, es decir, entre más oscuro es el nodo, quiere decir que tiene una centralidad de cercanía mayor, por lo que por ese nodo, suelen pasar más caminos cortos.



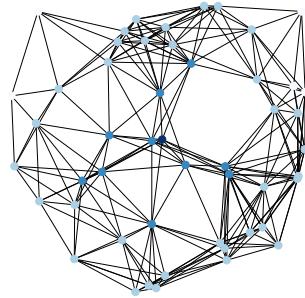
(a) Grafo 1

(b) Grafo 2



(c) Grafo 3

(d) Grafo 4



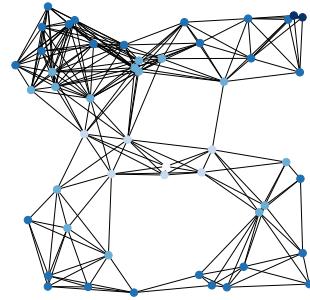
(e) Grafo 5

Figura 4: Centralidad de cercanía

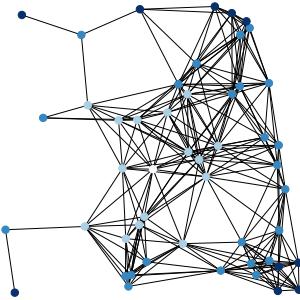
## Exentricidad

La figura 5 colorea los nodos dependiendo su valor de exentricidad, es decir, entre más oscuro es el nodo, quiere decir que tiene una exentricidad mayor, por lo que por ese nodo, pasan las

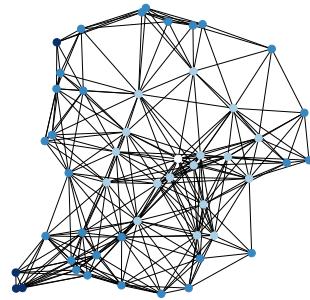
distancias máximas más grandes.



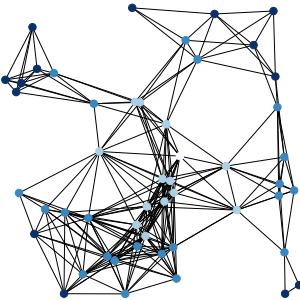
(a) Grafo 1



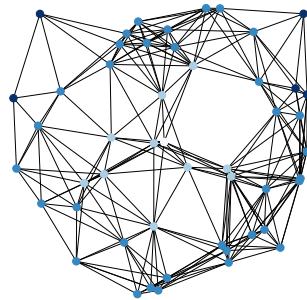
(b) Grafo 2



(c) Grafo 3



(d) Grafo 4

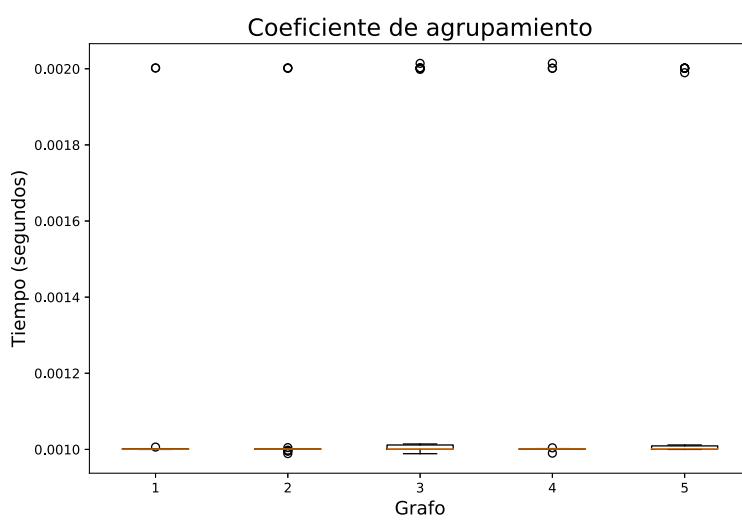
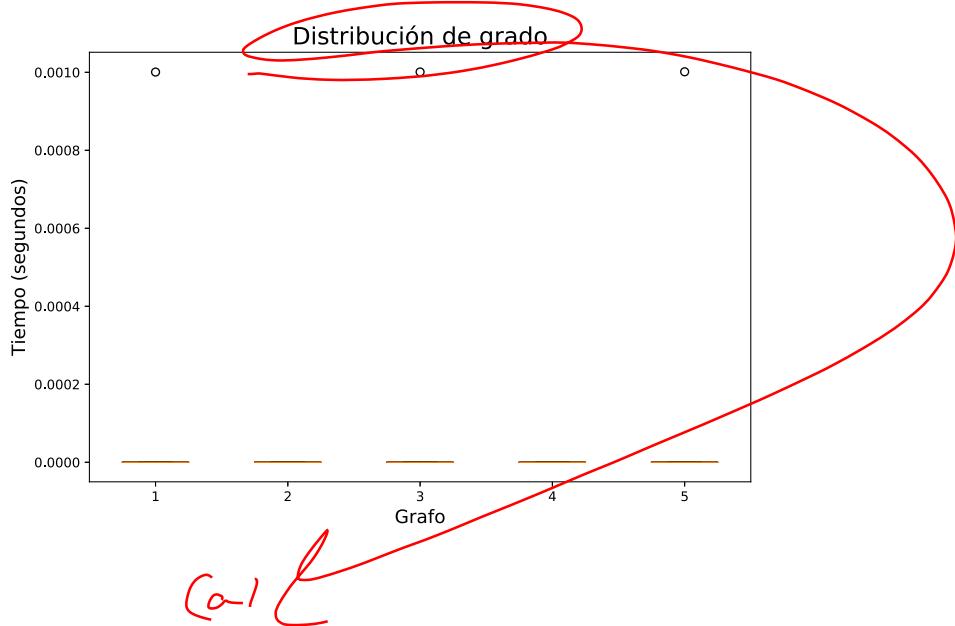


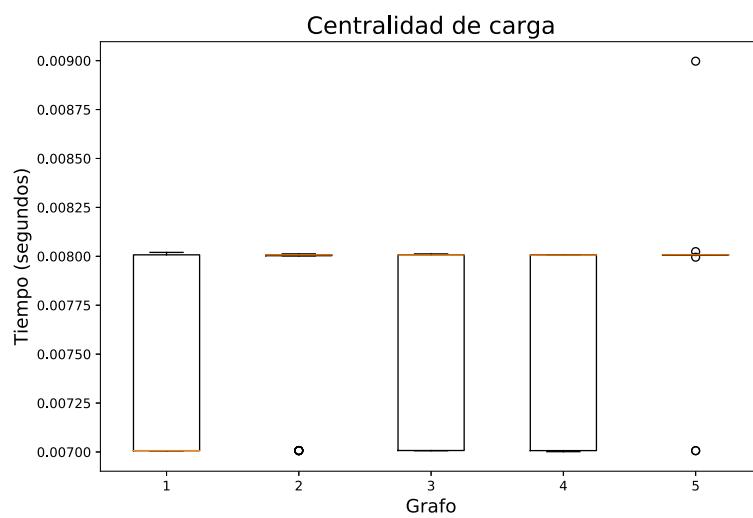
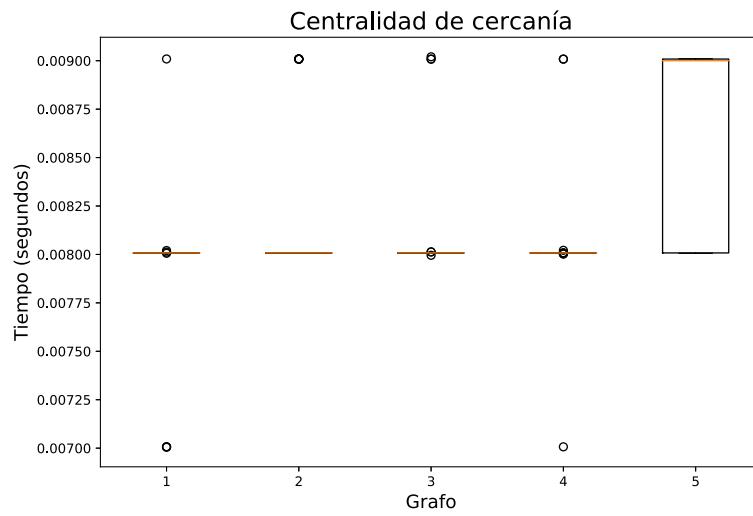
(e) Grafo 5

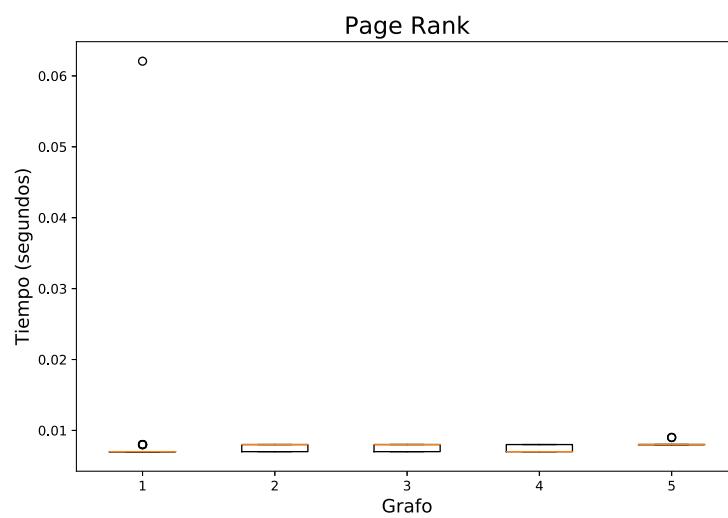
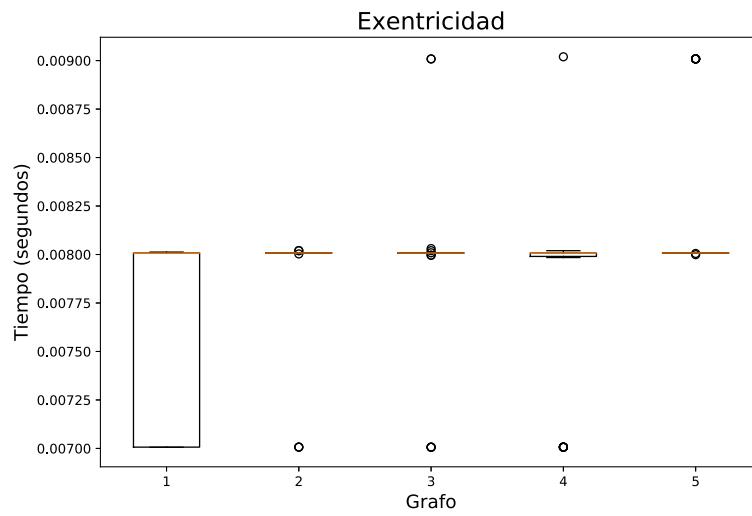
Figura 5: Exentricidad

A continuación se muestran los diagramas de caja y bigotes para los tiempos individuales de los cinco grafos utilizados al ejecutar las seis características estructurales.

*L* [ ]







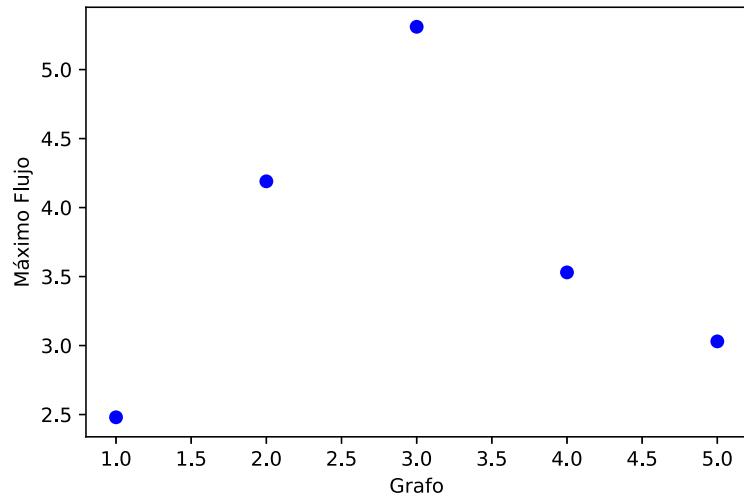


Figura 6: Grafos contra Valor óptimo

---

ANOVA.txt

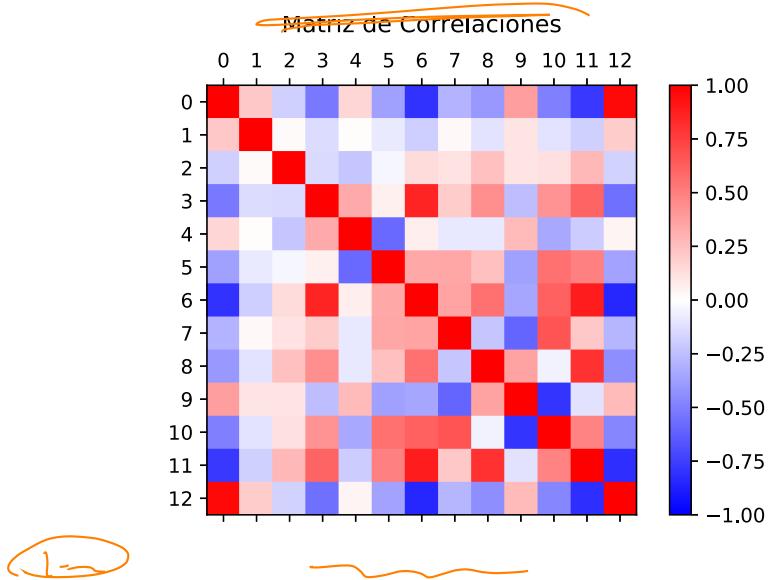
---

	sum_sq	...	PR>F
Clustering_Fuente	24505.051884	...	1.725271e-05
Load_Fuente	15446.314551	...	5.535417e-04
Closeness_Fuente	412.565629	...	5.643372e-01
Eccentricity_Fuente	1164.600442	...	3.333495e-01
PageRank_Fuente	6902.541440	...	1.949432e-02
Clustering_Sumidero	23527.493943	...	2.480762e-05
Load_Sumidero	153461.288835	...	5.263273e-21
Closeness_Sumidero	105337.759194	...	3.980059e-16
Eccentricity_Sumidero	266388.228363	...	4.696434e-30
PageRank_Sumidero	93111.247075	...	9.511706e-15
Residual	171775.004303	...	NaN

---

c@

---



## Conclusiones

De los diagramas de caja y bigote se puede apreciar estas características de los nodos no afectan al tiempo de ejecución del algoritmo seleccionado, sin embargo al ver la figura 6, se puede apreciar que al valor del óptimo si.

Si vemos las características en los nodos seleccionados como fuente y sumidero podemos ver que los nodos que cumplen las características del nodo fuente y nodo sumidero del grafo 3, resultan ser buenos nodos fuentes y buenos sumideros, ya que el valór del flujo máximo es mayor. Por otro, los nodos que cumplen las características del nodo fuente y nodo sumidero del grafo 1, sería mejor no usar como ninguno si uno busca obtener un alto flujo. Para ambos casos de nodos fuente y sumidero el tiempo de ejecución del algoritmo es rápido.

## Referencias

- [1] E. Schaeffer. Optimización de flujo en redes, 2019. <https://elisa.dyndns-web.com/teaching/opt/flow/>. Accedido el 2019-04-29.
- [2] A. Serna. Optimización de flujo en redes, 2019. <https://github.com/SernArmando>. Accedido el 2019-04-29.
- [3] Research Frontiers. Random geometric graphs and their applications, 2010. [https://www.ugc.edu.hk/minisite/rgc\\_newsletter/rgcnews18/eng/05.htm](https://www.ugc.edu.hk/minisite/rgc_newsletter/rgcnews18/eng/05.htm). Accedido el 2019-04-29.
- [4] NetworkX. Algorithms. <https://networkx.github.io/documentation/networkx-2.2/reference/algorithms/index.html>. Accedido el 2019-04-29.

# Caracterización de la red de calidad del aire en el Área Metropolitana de Monterrey

Matrícula: 1985281

---

## Resumen

Recientemente, han sucedido alarmas ambientales decretadas en Monterrey sobre la mala calidad del aire, que posiciona a la ciudad como una de las más contaminadas del país, en lo que se refiere a material particulado, como el *PM10* y el *PM2.5*. Este trabajo está enfocado en el problema de identificar, analizar y proponer un modelo que caracterice a la red de calidad del aire en el Área Metropolitana de Monterrey (AMM), con el fin de entender el comportamiento y evolución de esta.

---

## Introducción

El AMM cuenta con trece estaciones fijas de monitoreo bajo el encargo del Sistema Integral de Monitoreo Ambiental de la Secretaría de Desarrollo Sustentable (SIMA) del Gobierno de Nuevo León, ver figura 1. Los datos registrados por las estaciones de monitoreo dan una indicación de las tendencias en la calidad del aire en el AMM. En este trabajo se utilizan los datos promedios reportados del contaminante *PM10* por hora de todos los días del mes de noviembre de 2017 y se predice para el mes siguiente utilizando el método de *Mínimos Cuadrados Ordinarios*. Luego se establece un sistema de ecuaciones lineales con los datos encontrados por el método y se calcula el flujo de contaminación de una estación a otra.

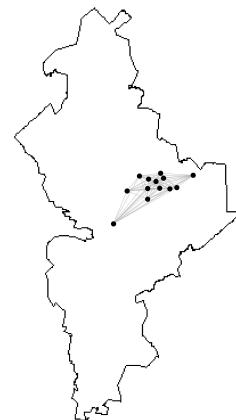
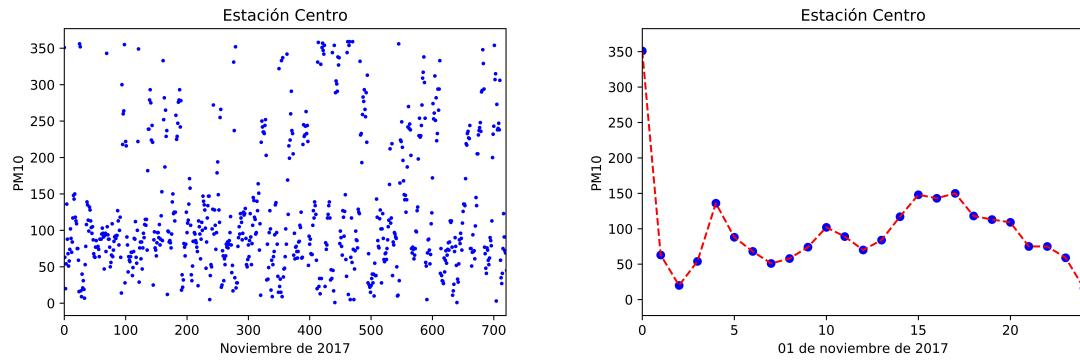


Figura 1: Representación de las trece estaciones de monitoreo en el AMM

## Metodología y Resultados

Se obtienen los datos del contaminante PM10 por cada estación de monitoreo, la figura 2 (a), representa los valores obtenidos por el contaminante PM10 durante todo el mes de noviembre en la estación Centro y la figura 2 (b), representa los valores obtenidos por el contaminante PM10 durante todo el día 01 de noviembre de 2017 en la misma estación.



(a) Registro de PM10 en la estación *Centro* durante noviembre de 2017 (b) Registro de PM10 en la estación *Centro* durante 01 de noviembre de 2017

Figura 2: Registro de PM10 en la estación *Centro*

Se construye la red completa de las trece estaciones, donde los nodos representan las posiciones de las estaciones de monitoreo y las aristas representan la interacción del contaminante PM10 entre éstos. A los nodos se les identifica con un color dependiendo del nivel de contaminación que este reportando actualmente la estación de monitoreo.

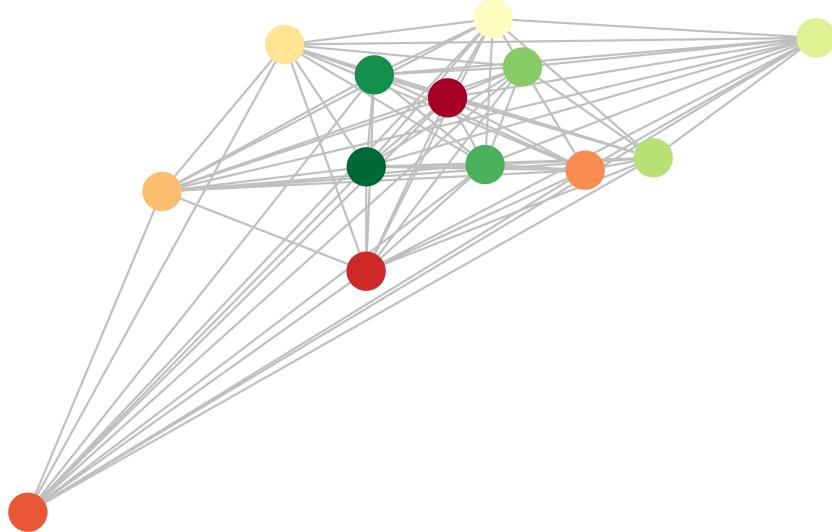


Figura 3: Representación de las trece estaciones de monitoreo en el AMM al 01 de noviembre de 2017 a las 00:00:00 hrs.

En la figura 3 se puede apreciar que al 01 de noviembre del 2017 la estación que reporta mayor contaminación en lo que respecta a PM10 es la estación centro, mientras que las estaciones alrededor de ella presentan registros menores de contaminante PM10 a las 00:00:00 hrs.

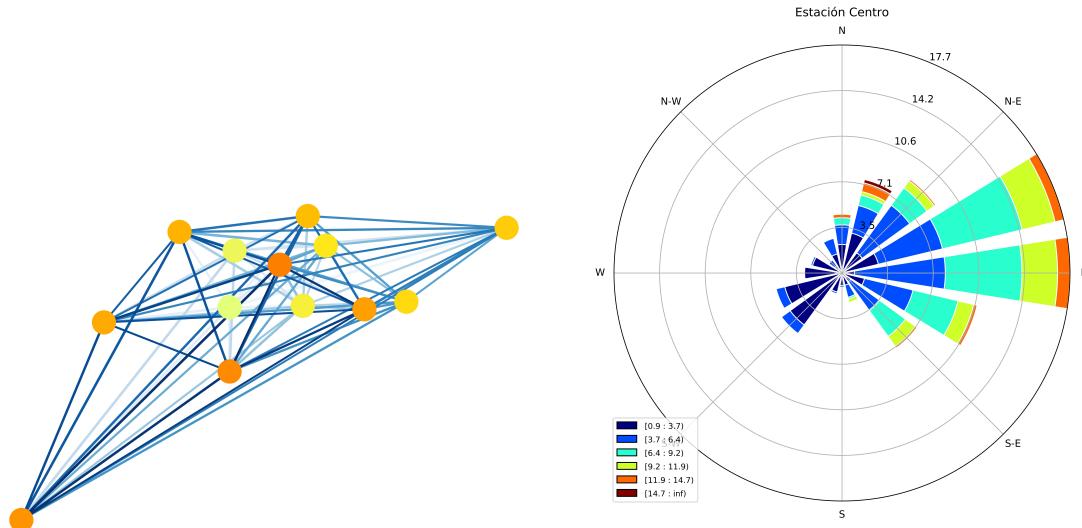
Con el método de mínimos cuadrados ordinarios se calculan los estimadores de la siguiente hora del día 01 de diciembre de 2017 (00:00:00 hrs) para cada estación de la red de la variable de respuesta del contaminante PM10 ( $e_i$ ), considerando la velocidad y la dirección del viento como variables explicativas, ver la implementación en el código para la estación Centro.

```

1 centro = por_estacion[0]
2 result = sm.ols(formula="PM10 ~ velocity + direction", data=centro).fit()
3 respuesta_centro = result.params[0]
```

Se calculan los pesos de las aristas ( $x_{ij}$ ) considerando, los valores estimados ( $e_i$ ), las distancias entre las estaciones ( $d_{ij}$ ), la velocidad del viento ( $v_i$ ) y las direcciones del viento ( $\alpha_i$ ), de la siguiente manera:

$$e_i(t+1) = \sum_{j \neq i} \left( \left( \frac{1}{d_{ij}(t)} v_i(t) \right) + \alpha_i(t) \right) x_{ij}(t+1), \quad i, j = 1, \dots, 13.$$



(a) Calidad del aire por estación al 01 de diciembre de 2017 a las 00:00:00 hrs. (b) Rosa de viento durante noviembre de 2017 de la estación Centro

Figura 4: Registro de PM10 por estación

En la figura 4 (a) se observa la calidad del aire en el AMM, obtenida con los valores estimados al día 01 de diciembre de 2017 a las 00:00:00 hrs. y la 4 (b) srepresenta la rosa de viento de la estación centro, la cual explica la dirección y velocidad del aire en esa estación.

## Conclusiones

Se puede concluir que la medición del contaminante PM10, esta relacionado con la velocidad y la dirección del viento, por ejemplo, en la figura 4 (b) se observa que la dirección del viento corre

de este a oeste, sin embargo la velocidad con la que llega a ese punto es menor que con la que sale, esto indica que hay propiedades topológicas que modifican la valocidad del viento haciendo que sea más lento y que haya una mayor concentración de particulas PM10 en la estación Centro. Al comparar con lo obtenido por los estimadores podemos ver en la figura 4 (a) que las estaciones al este de la estación centro reportan una menor contaminación de PM10, lo que quiere decir que el viento es causante de la interacción entre contaminantes de la red, mientras que la estación Centro reporta un alto registro de particulas PM10. Támbien se puede observar que las estaciones al oeste de la estación Centro reportan aún menores índices de contaminante que la estación Centro lo que indica que el principal factor es la velocidad del aire.

## Referencias

- [1] E. Schaeffer. Optimización de flujo en redes, 2019. <https://elisa.dyndns-web.com/teaching/opt/flow/>. Accedido el 2019-06-01.
- [2] A. Serna. Optimización de flujo en redes, 2019. <https://github.com/SernArmando>. Accedido el 2019-06-03.
- [3] Research Frontiers. Random geometric graphs and their applications, 2010. [https://www.ugc.edu.hk/minisite/rgc\\_newsletter/rgcnews18/eng/05.htm](https://www.ugc.edu.hk/minisite/rgc_newsletter/rgcnews18/eng/05.htm). Accedido el 2019-06-01.
- [4] NetworkX. Algorithms. <https://networkx.github.io/documentation/networkx-2.2/reference/algorithms/index.html>. Accedido el 2019-06-02.

## Anexos

### Práctica 5

Se dibujaron otra vez los grafos instancias, ya que en la Practica 5 el grosor del contorno de las aristas no de distinguia adecuadamente.

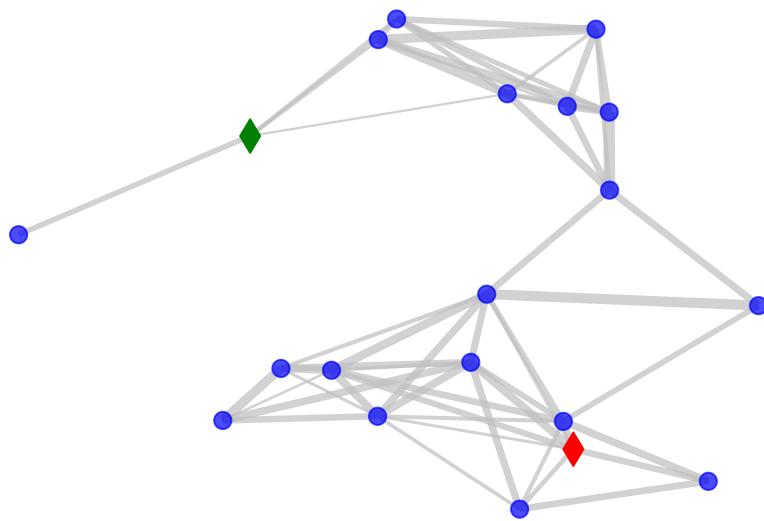


Figura 1: Grafo 1

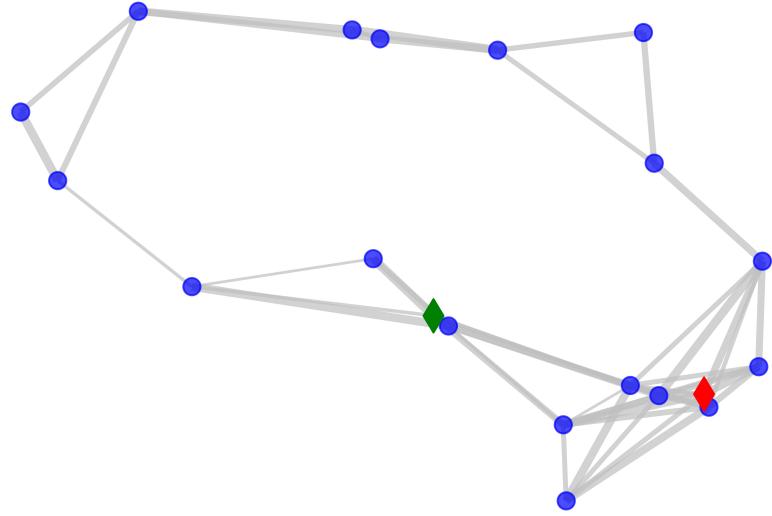


Figura 2: Grafo 2

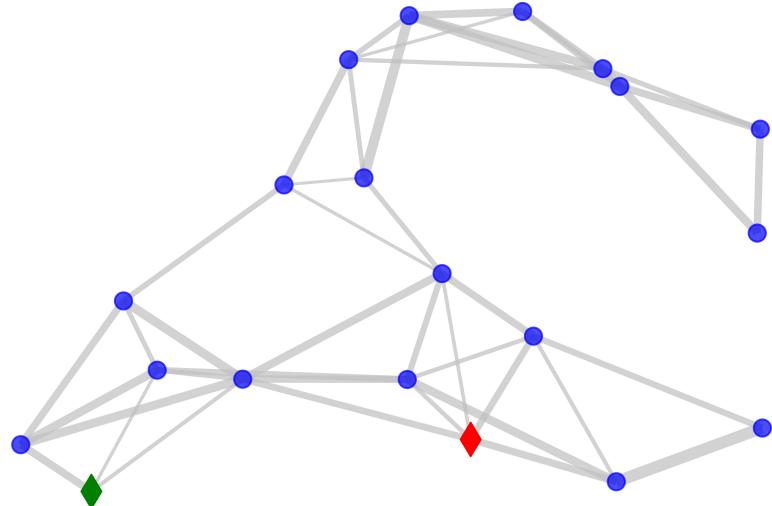


Figura 3: Grafo 3

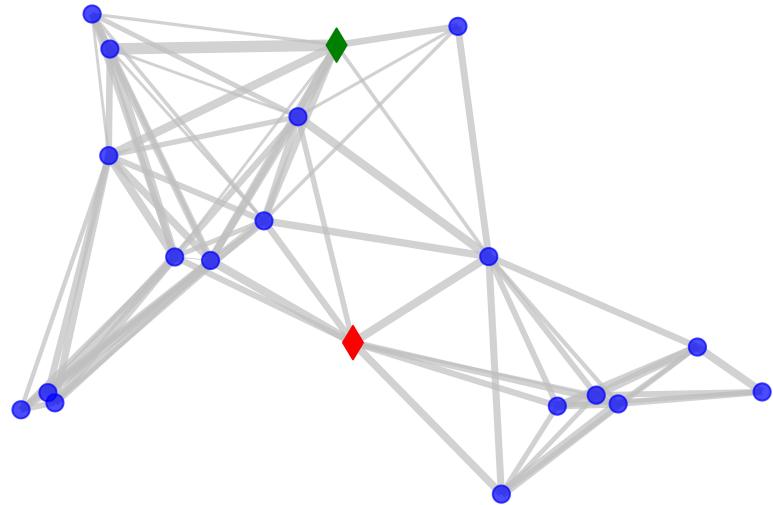


Figura 4: Grafo 4

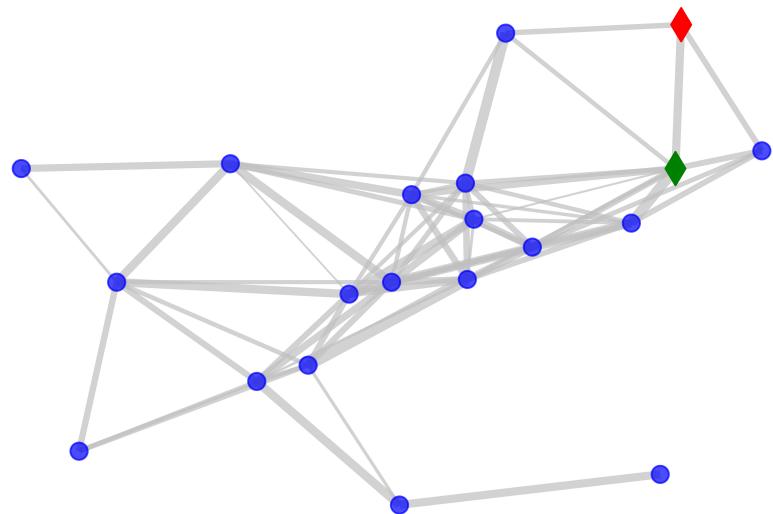


Figura 5: Grafo 5