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 *Fruchterman-Reingold*
shell_layout	En círculos concéntricos
fruchterman_reingold_layout	Utiliza el algoritmo *Fruchterman-Reingold*
kamada_kawai_layout	Utiliza la función *Kamada-Kawai*

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 escoje 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

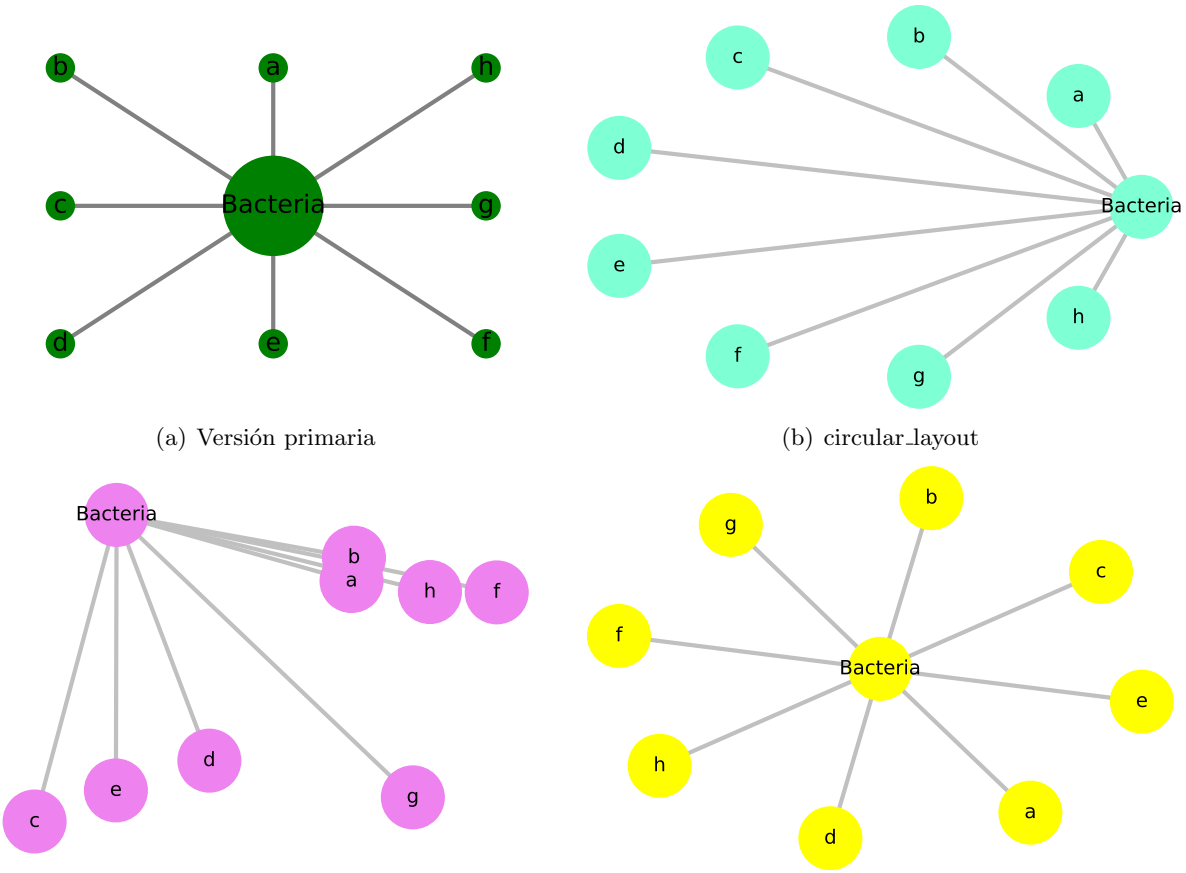
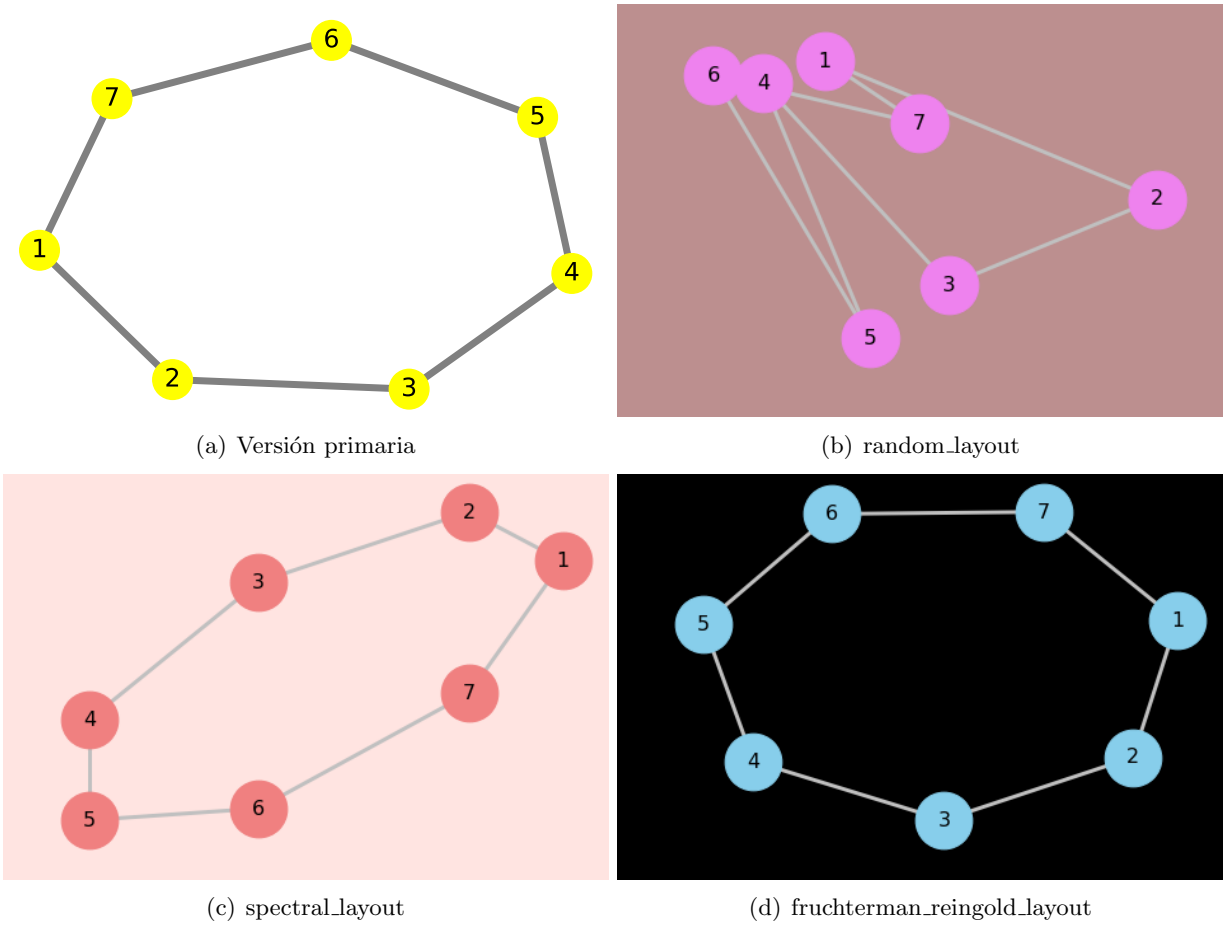


Figura 1: Ejemplo de la Filogenia

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 démas 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



(a) Versión primaria

(b) random_layout

(c) spectral_layout

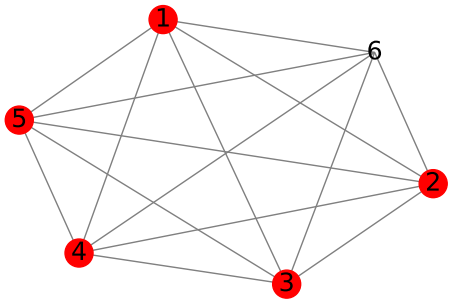
(d) fruchterman_reingold_layout

Figura 2: Ejemplo del problema de TSP

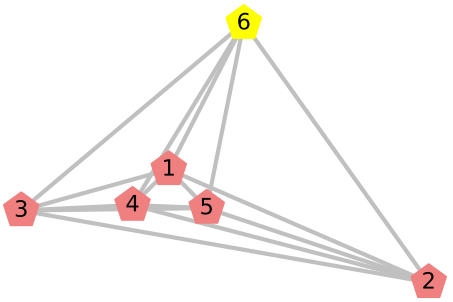
En la figura 2 se puede apreciar que el algoritmo de acomodo que mejor dibuja la red es el algoritmo que da por default NetworkX, ya que por ejemplo, el algoritmo `random_layout` no genera un dibujo del grafo bueno visualmente. A continuación se muestra parte del código en Python para dibujar grafos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

```
fig = plt.figure()
nx.draw(G, with_labels=True, node_size=2000, node_color="violet", pos=nx.
    random_layout(G), width=3, edge_color='silver', font_size=16)
fig.set_facecolor("rosybrown")

fig = plt.figure()
nx.draw(G, with_labels=True, node_size=2000, node_color="lightcoral", pos=nx.
    spectral_layout(G), width=3, edge_color='silver', font_size=16)
fig.set_facecolor("mistyrose")

fig = plt.figure()
nx.draw(G, with_labels=True, node_size=2000, node_color="skyblue", pos=nx.
    fruchterman_reingold_layout(G), width=3, edge_color='silver', font_size=16)
fig.set_facecolor("black")
```

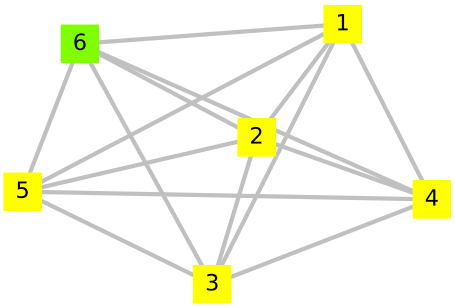
Grafo simple no dirigido reflexivo



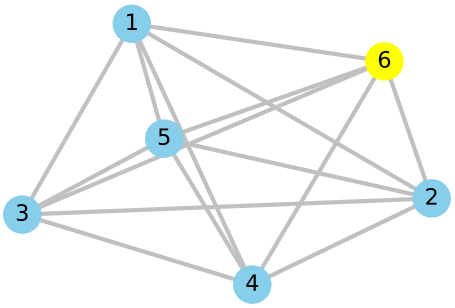
(a) Versión primaria



(b) spectral_layout



(c) spring_layout



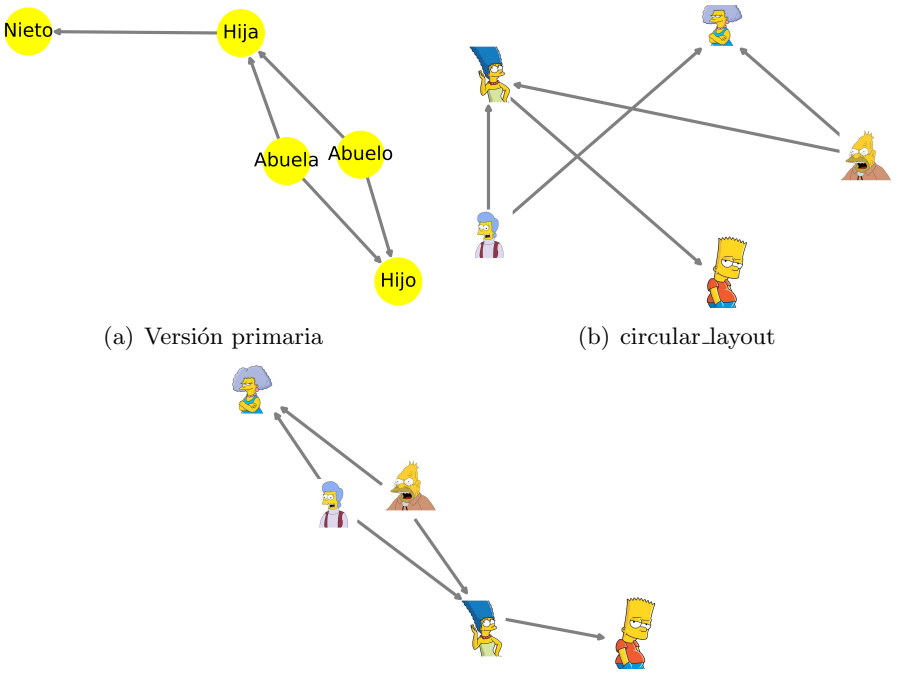
(d) fruchterman_reingold_layout

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.

```
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)

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)

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



(a) Versión primaria

(b) circular_layout

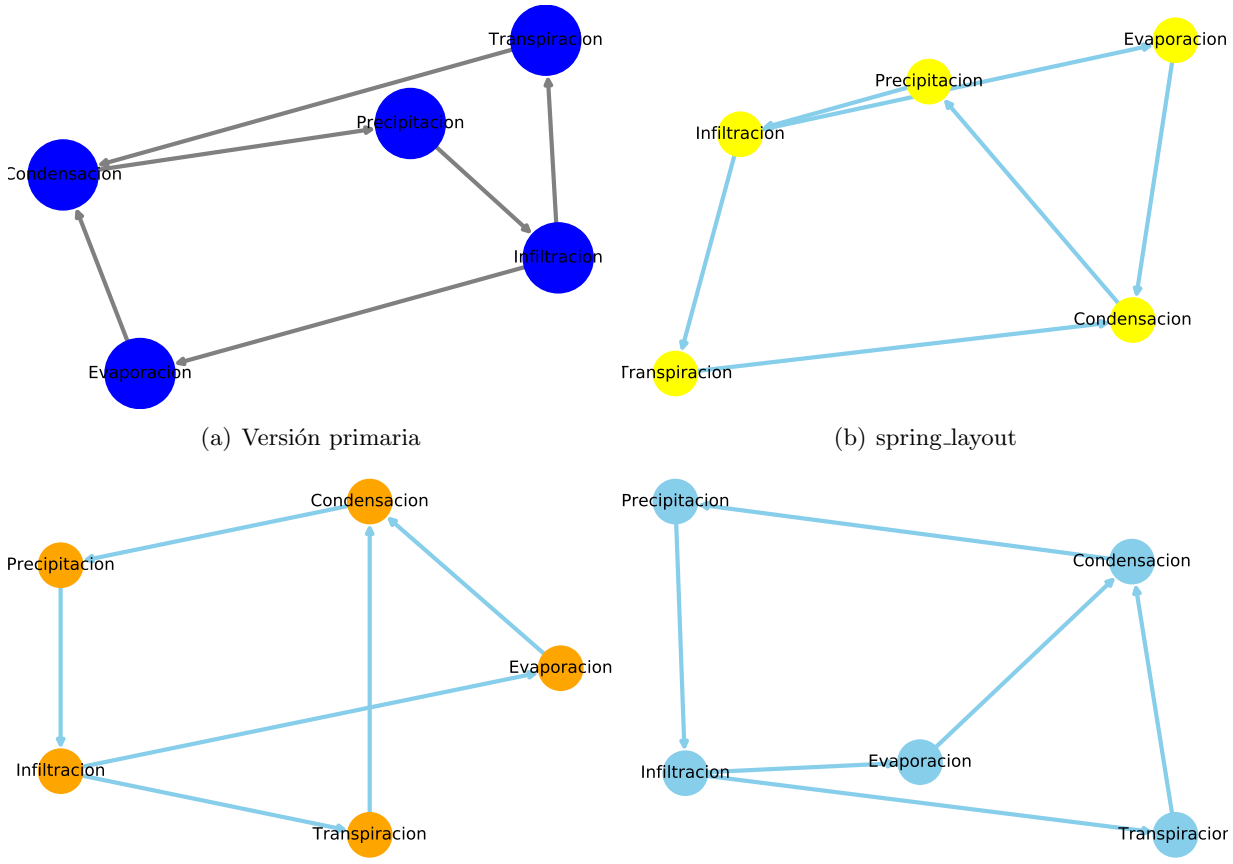
(c) spring_layout

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].

```
from PIL import Image
def main():
    try:
        img = Image.open("grafo4.png")
        img1 = Image.open("abuelo.png")
        img2 = Image.open("abuela.png")
        img3 = Image.open("mama.png")
        img4 = Image.open("hermana.png")
        img5 = Image.open("hijo.png")
        img.paste(img1, (2200,1250))
        img.paste(img2, (1200,1500))
        img.paste(img3, (3200,3100))
        img.paste(img4, (100,0))
        img.paste(img5, (5200,3000))
    except IOError:
        pass
if __name__ == "__main__":
    main()
```

Grafo simple dirigido cíclico



(a) Versión primaria

(b) spring_layout

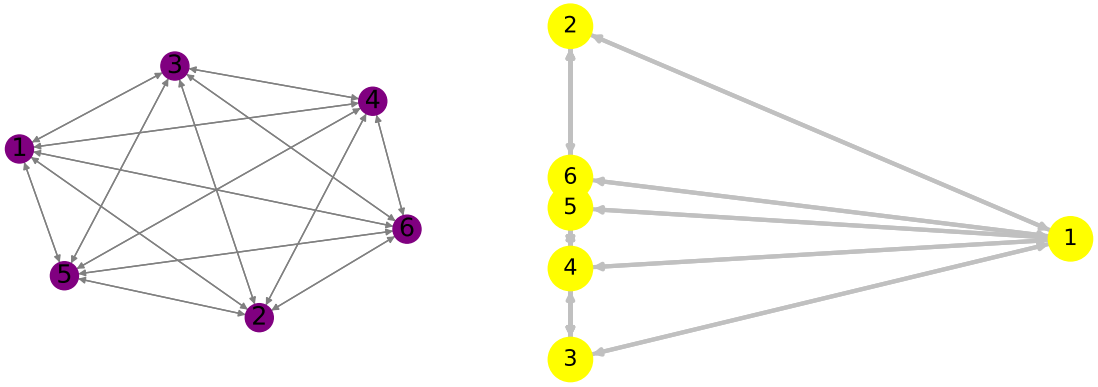
(c) shell_layout

(d) fruchterman_reingold_layout

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

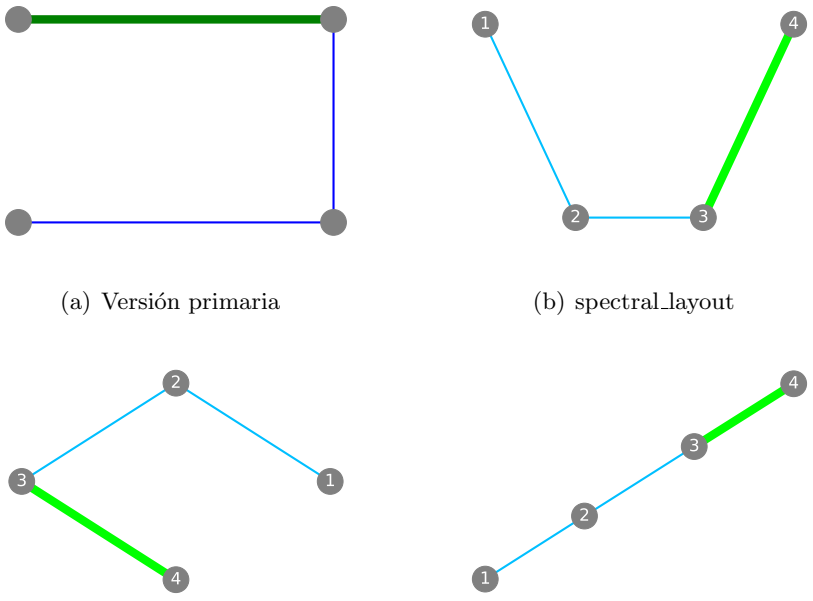


(a) Versión primaria (b) spectral_layout

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 NetworkXy el algoritmo de acomodo `spectral_layout`, ya que ambos algorimos generan dibujos del gráfico buenos visualmente.

Multigrafo no dirigido acíclico



(a) Versión primaria (b) spectral_layout

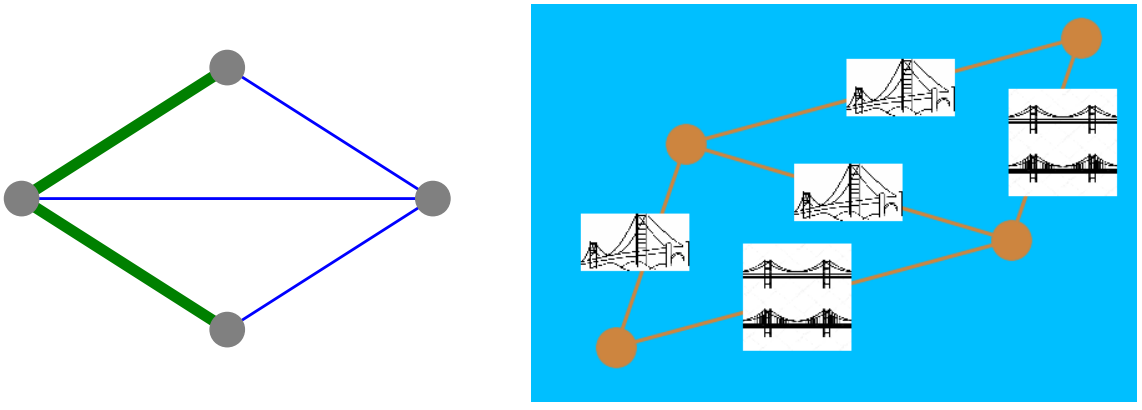
(c) shell_layout (d) fruchterman_reingold_layout

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.

```
pos=nx.spectral_layout(G)
nx.draw_networkx_nodes(G, pos, node_color='grey', node_size=600)
nx.draw_networkx_edges(G, pos, edgelist=[(1,2), (2,3)], width=2, edge_color='
    deepskyblue')
nx.draw_networkx_edges(G, pos, edgelist=[(3, 4)], width=8, edge_color='lime')
nx.draw_networkx_labels(G, pos, font_size=16, font_color='white')

pos=nx.shell_layout(G)
nx.draw_networkx_nodes(G, pos, node_color='grey', node_size=600)
nx.draw_networkx_edges(G, pos, edgelist=[(1,2), (2,3)], width=2, edge_color='
    deepskyblue')
nx.draw_networkx_edges(G, pos, edgelist=[(3, 4)], width=8, edge_color='lime')
nx.draw_networkx_labels(G, pos, font_size=16, font_color='white')

pos=nx.fruchterman_reingold_layout(G)
nx.draw_networkx_nodes(G, pos, node_color='grey', node_size=600)
nx.draw_networkx_edges(G, pos, edgelist=[(1,2), (2,3)], width=2, edge_color='
    deepskyblue')
nx.draw_networkx_edges(G, pos, edgelist=[(3, 4)], width=8, edge_color='lime')
nx.draw_networkx_labels(G, pos, font_size=16, font_color='white')
```

Multigrafo no dirigido cíclico



(a) Versión primaria

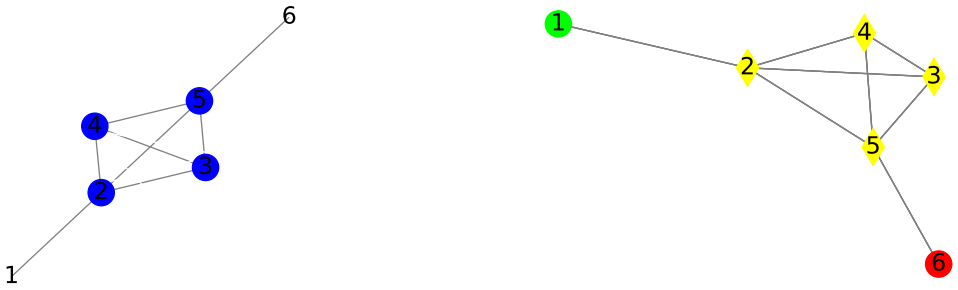
(b) spring_layout

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].

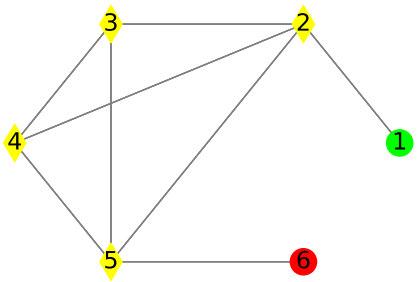
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
pos=nx.spring_layout(G)
ax=plt.gca()
fig=plt.gcf()
label_pos = 0.5
trans = ax.transData.transform
trans2 = fig.transFigure.inverted().transform
imsize = 0.1
for (n1,n2) in G.edges():
    (x1,y1) = pos[n1]
    (x2,y2) = pos[n2]
    (x,y) = (x1 * label_pos + x2 * (1.0 - label_pos),
             y1 * label_pos + y2 * (1.0 - label_pos))
    xx,yy = trans((x,y))
    xa,ya = trans2((xx,yy))
    imsize = G[n1][n2]['size']
    img =  G[n1][n2]['image']
    a = plt.axes([xa-imsize/2.0,ya-imsize/2.0, imsize, imsize ])
    a.imshow(img)
```

Multigrafo no dirigido reflexivo

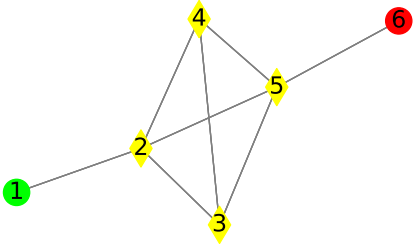


(a) Versión primaria

(b) spring_layout



(c) shell_layout

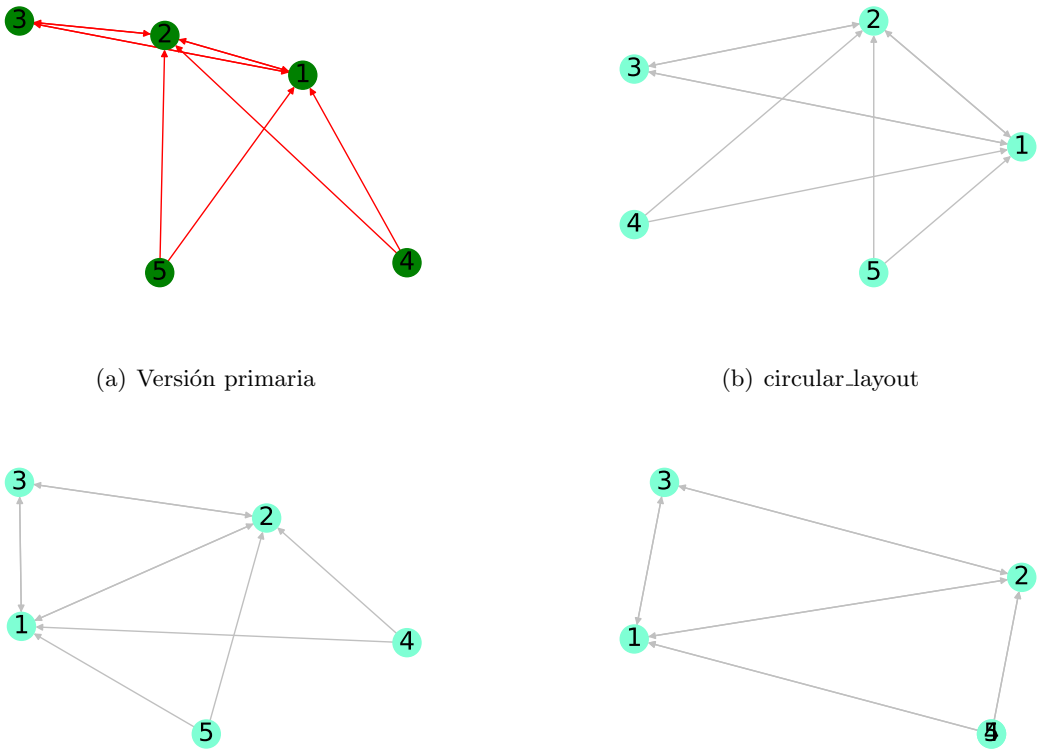


(d) kamada_kawai_layout

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

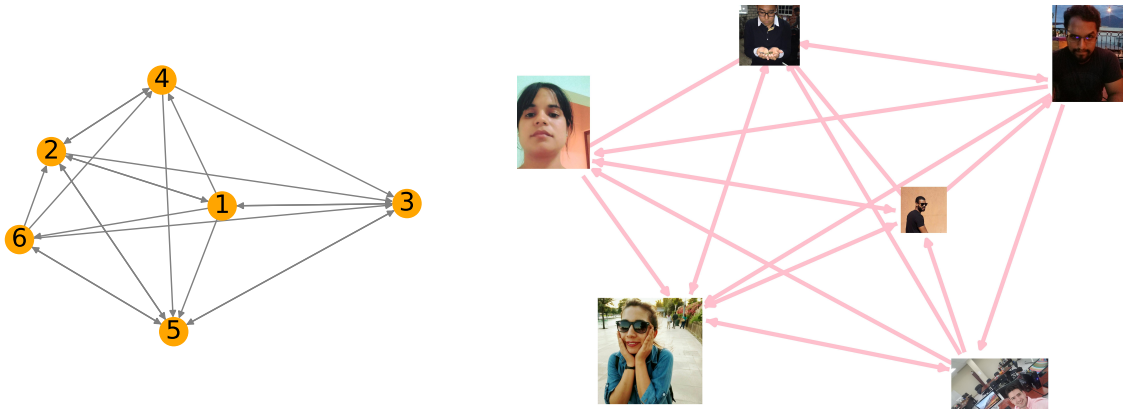
(c) fruchterman_reingold_layout

(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 NetworkXy el algoritmo `kamada_kawai_layout`, no generan un dibujo del grafo bueno visualmente.

Multigrafo dirigido cíclico

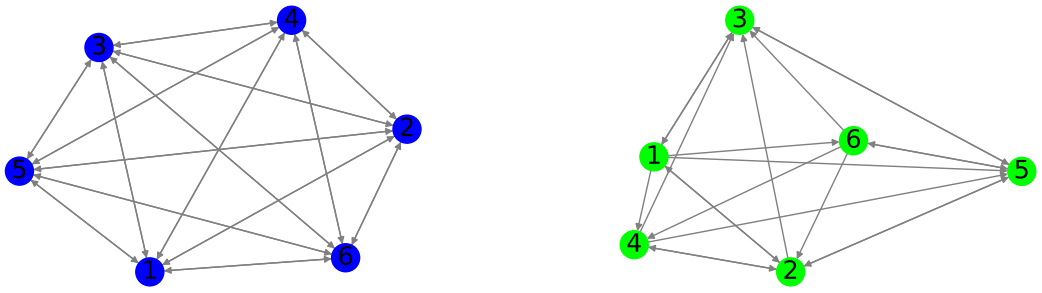


(a) Versión primaria (b) circular_layout

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



(a) Versión primaria (b) kamada_kawai_layout

Figura 12: Ejemplo de transmición 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 Applicattions. *[Prentice Hall]*, 1993.

[2] M. Bazaraa, J. Jarvis, and H. Sherali. *Linear Programming and Netwok 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/.