Diseño de los grafos

1484166

Introducción

Un grafo se puede dibujar de muchas maneras ya que por lo que representan aveces es conveniente verlo de cierto modo. En este documento se muestra algunas opciones presentes en la librería de `Networkx`

1. Diseño bipartito

Un grafo bipartito es aquel donde sus nodos se pueden repartir en dos conjuntos disjuntos.

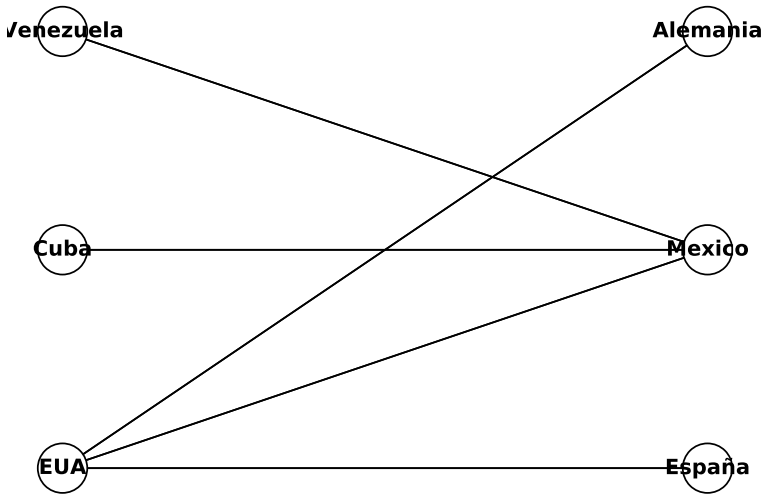
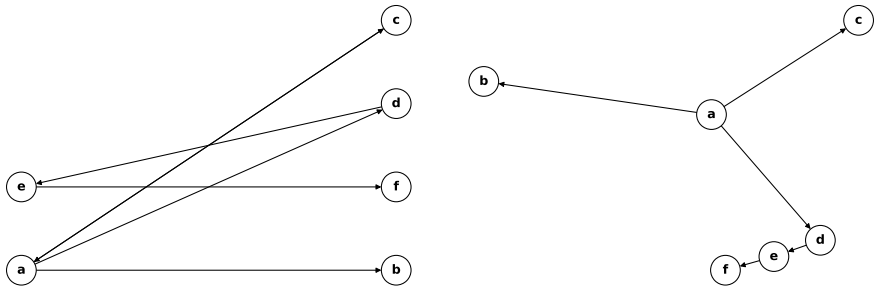


Figura 1: Grafo bipartito



(a) Grafo bipartito (b) Grafo dirigido acíclico

Figura 2: Comparativa entre grafo con acomodo bipartito `vs` grafo con acomodo por default

El código que da paso a la figura 1 es el siguiente.

```
H=nx.MultiGraph()
H.add_nodes_from(['Venezuela','Mexico','Cuba'], bipartite=0)
H.add_nodes_from(['Espa a','EUA','Alemania'], bipartite=1)
H.add_edges_from([('Mexico','EUA'), ('EUA','Mexico'),
                  ('EUA','Espa a'), ('Espa a','EUA'),
                  ('Mexico','Venezuela'), ('Venezuela','Mexico'),
                  ('Cuba','Mexico'), ('Mexico','Cuba'),
                  ('EUA','Alemania'), ('Alemania','EUA')])
X,Y=bipartite.sets(H)
pos=dict()
pos.update( (n, (1, i)) for i, n in enumerate(X) )
pos.update( (n, (2, i)) for i, n in enumerate(Y) )
nx.draw(H, pos, node_color="white", node_size=800, with_labels=True
    ,
        font_weight="bold", edgecolors="black")
plt.savefig('SSeptimo.eps', format='eps', dpi=1000)
```

Grafos_layout.py

2. Diseño circular

Como lo dice su nombre, un grafo de este tipo es el que tiene una distribución circular. Para hacer estos grafos se usa la función `circular_layout()`

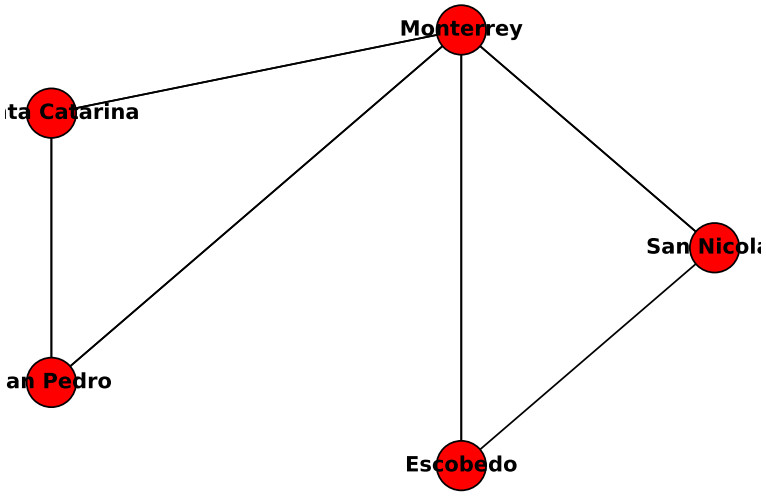
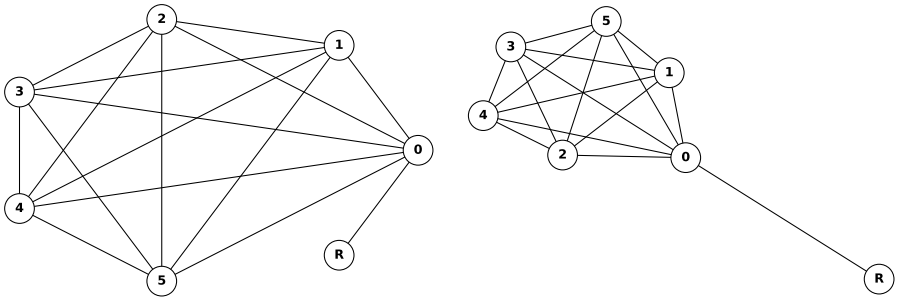


Figura 3: Grafo circular



(a) Grafo circular (b) Grafo no dirigido ciclico

Figura 4: Comparativa entre grafo con acomodo circular `vs` grafo acomodo por default

Para graficar la red de la figura 4, grafo con red circular se utilizó el siguiente código.

```
H=nx.complete_graph(6)
H.add_nodes_from([1,2,3,4,5])
H.add_edge(0,'R')
pos = nx.circular_layout(H)
nx.draw(H, pos, node_color="white", node_size=800, with_labels=True,
        font_weight="bold", edgecolors="black")
plt.savefig('SSegundo.eps', format='eps', dpi=1000)
```

Grafos_layout.py

3. Diseño spectral

En este acomodo del grafo se utiliza los vectores propios de una matriz como coordenadas cartesianas de los nodos de la gráfica. Para acceder a este diseño de grafos basta con ingresar la función `spectral_layout()`.

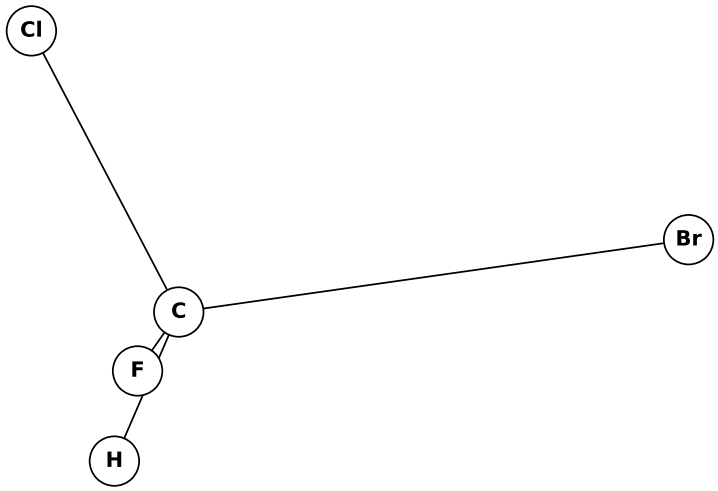
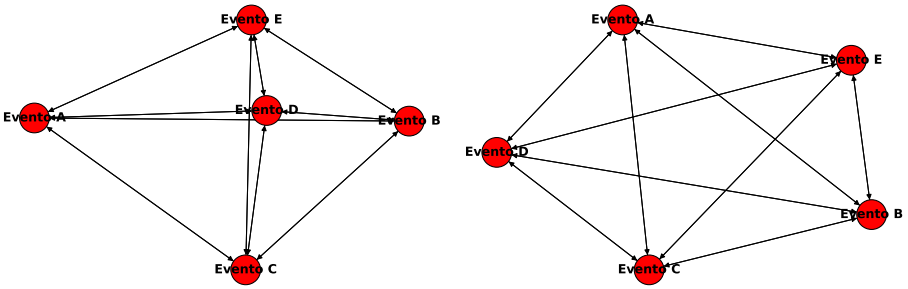


Figura 5: Grafo spectral



(a) Grafo spectral

(b) Multigrafo dirigido reflexivo

Figura 6: Comparativa entre grafo con acomodo espectral `vs` grafo con acomodo por default

El código con el que se hizo el grafo de la figura 6 con acomodo espectral es.

```
G=nx.MultiDiGraph()
G.add_weighted_edges_from([('Evento A','Evento B',1),
                           ('Evento B','Evento A',2),
                           ('Evento A','Evento C',3),
                           ('Evento C','Evento A',2),
                           ('Evento A','Evento D',4),
                           ('Evento D','Evento A',1),
                           ('Evento A','Evento E',3),
                           ('Evento E','Evento A',2),
                           ('Evento B','Evento C',4),
                           ('Evento C','Evento B',1),
                           ('Evento B','Evento D',5),
                           ('Evento D','Evento B',1),
                           ('Evento B','Evento E',2),
                           ('Evento E','Evento B',3),
                           ('Evento C','Evento D',4),
                           ('Evento D','Evento C',2),
                           ('Evento C','Evento E',3),
                           ('Evento E','Evento C',1),
                           ('Evento D','Evento E',5),
                           ('Evento E','Evento D',2)])
pos=nx.spectral_layout(G)
nx.draw(G, pos, node_size=800, with_labels=True,
        font_weight="bold", edgecolors="black", node_color='red')
plt.savefig('DDoceavo.eps', format='eps', dpi=1000)
```

Grafos_layout.py

4. Diseño spring

Esta función hace una analogía de las fuerzas de física, el objetivo de este algoritmo es encontrar un acomodo de los nodos donde se minimice la suma de las fuerzas. Para obtener este tipo de grafo se utiliza la función `spring_layout()`. Algunos ejemplos de muestran a continuación.

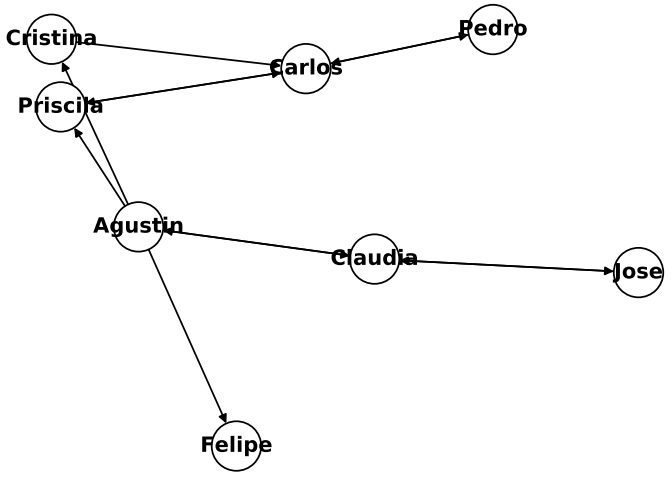
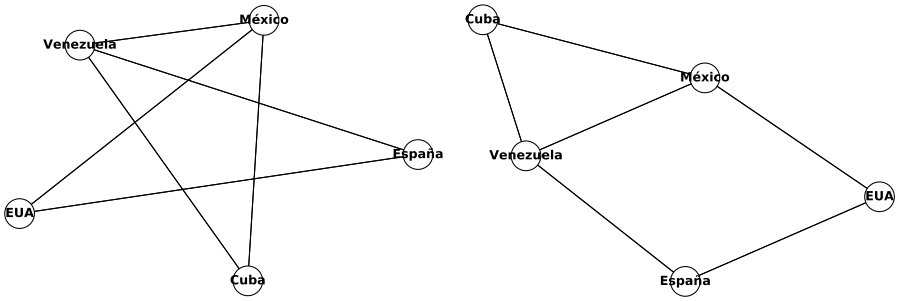


Figura 7: Grafo spring



(a) Grafo spring

(b) Multigrafo no dirigido ciclico

Figura 8: Comparativa entre grafo con acomodo spring `vs` grafo con acomodo por default

El siguiente código muestra como se hace el grafo de la figura 8.

```
H=nx.MultiGraph()
H.add_edges_from([('México','EUA'), ('EUA','México'),('EUA','España'),
                  ('España','EUA'), ('México','Venezuela'), ('Venezuela','México'),
                  ('Cuba','México'), ('México','Cuba'),('Cuba','Venezuela'), ('Venezuela','Cuba'),
                  ('Venezuela','España'), ('España','Venezuela')])
pos=nx.spring_layout(H)
nx.draw(H, pos, node_color="white", node_size=800, with_labels=True,
        font_weight="bold", edgecolors="black")
plt.savefig('OOctavo.eps', format='eps', dpi=1000)
```

Grafos_layout.py

5. Diseño Kamada Kawai

Este tipo diseño se utiliza cuando se quiere reducir el número de cruces de los aristas, como se muestra en los siguientes gráficos. Para hacer uso de este diseño se usa la función `kamada_kawai_layout()`.

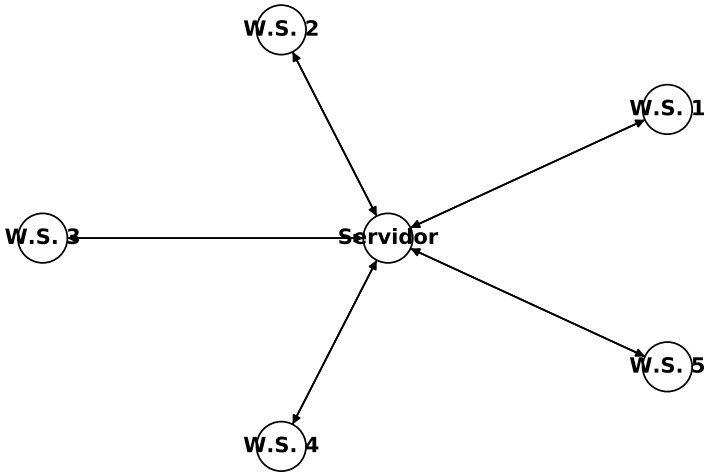
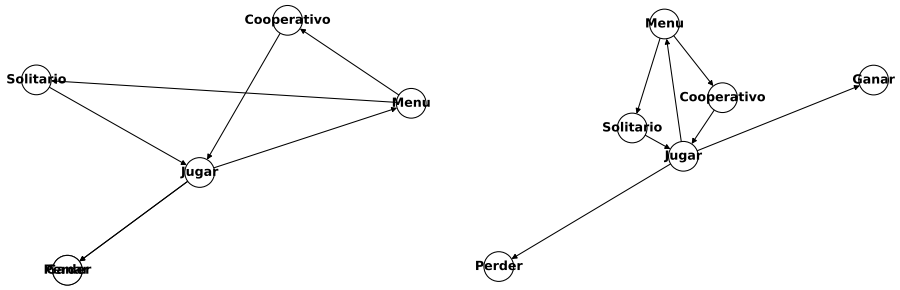


Figura 9: Grafo con diseño Kamada Kawai



(a) Grafo diseño Kamada Kawai (b) Grafo dirigido cíclico

Figura 10: Comparativa entre grafo con acomodo Kamada Kawai `vs` grafo acomodo por default

El código con el que se hizo la red de la figura 10 con el acomodo Kamada Kawai es.

```
H=nx.DiGraph()
H.add_edges_from([('Menu','Cooperativo'),('Menu','Solitario'),
                  ('Solitario','Jugar'),('Cooperativo','Jugar'),
                  ('Jugar','Menu'),('Jugar','Ganar'),
                  ('Jugar','Perder')])
pos=nx.kamada_kawai_layout(H)
nx.draw(H, pos, node_color="white", node_size=800, with_labels=True
    ,
        font_weight="bold", edgecolors="black")
plt.savefig('QQuinto.eps', format='eps', dpi=1000)
```

Grafos_layout.py

6. Diseño shell

Este algoritmo se basa en acomodar los nodos en circulos concéntricos.

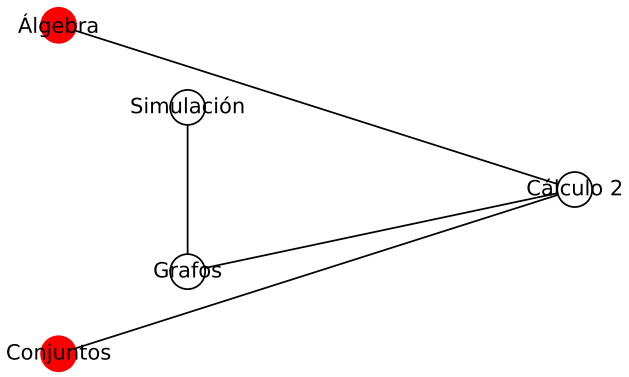
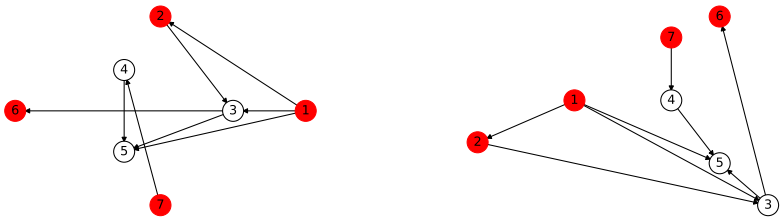


Figura 11: Grafo dirigido reflexivo



(a) Grafo diseño shell

(b) Grafo dirigido cíclico

Figura 12: Comparativa entre grafo con acomodo shell `vs` grafo acomodo por default

Para hacer la red de la figura 11 con el acomodo shell es.

```
G=nx.Graph()
G.add_edges_from([('  lgebra  ','C lculo 2'),('C lculo 2','Grafos')
    ,
                  ('Simulaci n','Grafos'),('C lculo 2','Conjuntos
    ')])

nodos1 = {'  lgebra  ','Conjuntos'}
nodos2 = {'C lculo 2','Simulaci n','Grafos'}

shells=[['C lculo 2','Simulaci n','Grafos'],['C lculo 1','
      lgebra  ',
         'Conjuntos']]

pos=nx.shell_layout(G,nlist=shells)

nx.draw_networkx_nodes(G, pos, nodelist=nodos1,node_size=400,
                       node_color='r', node_shape='o')
nx.draw_networkx_nodes(G, pos, nodelist=nodos2,node_size=400,
                       node_color='w', node_shape='o',
                       edgecolors="k")

nx.draw_networkx_edges(G, pos)
nx.draw_networkx_labels(G, pos)
plt.axis('off')

plt.savefig('TTercero.eps', format='eps', dpi=1000)
```

Grafos_layout.py

Referencias

[1] A. C. Barrero, G. W. de García, and R. M. M. Parra, *Introducción a la Teoría de Grafos*. ELIZCOM SAS, 2010.

[2] N. Developers, "Networkx documentation," 2012.

[3] J. M. Six and I. G. Tollis, "A framework for circular drawings of networks," in *International Symposium on Graph Drawing*. Springer, 1999, pp. 107–116.