Tarea 2

5171

26 de febrero de 2019

1. Algoritmo de acomodo Bipartito

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

Código

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

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

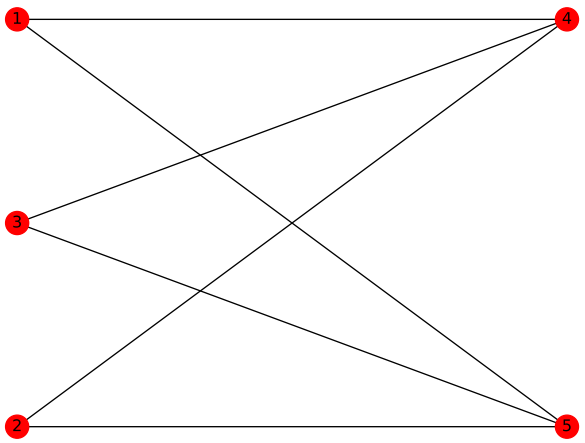

Figura 1: Grafo simple no dirigido acíclico

2. Algoritmo de acomodo circular

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

2.1. Código

```
import networkx as nx

import matplotlib.pyplot as plt

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


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

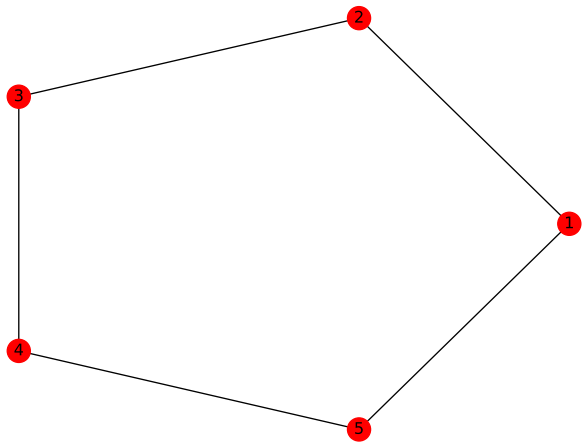

Figura 2: Grafo simple no dirigido cíclico

3. Algoritmo de acomodo kamada kawai

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

3.1. Código

```
from typing import Dict, Tuple

import networkx as nx

import matplotlib.pyplot as plt

```

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

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

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

plt.show()
```

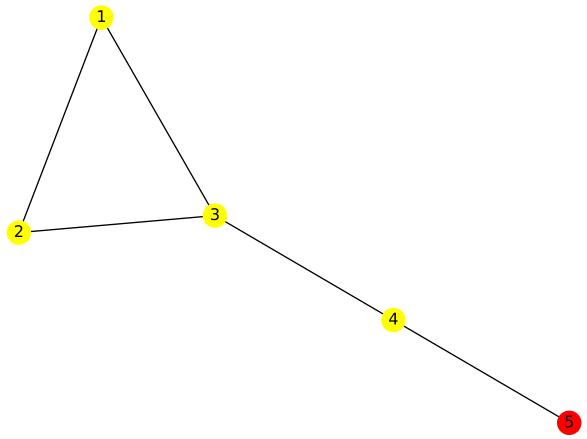

Figura 3: Grafo simple no dirigido reflexivo

4. Algoritmo de acomodo aleatorio.

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

4.1. Código

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

G=nx.DiGraph()

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

nx.draw_random(G,with_labels=True)

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

plt.show()
```

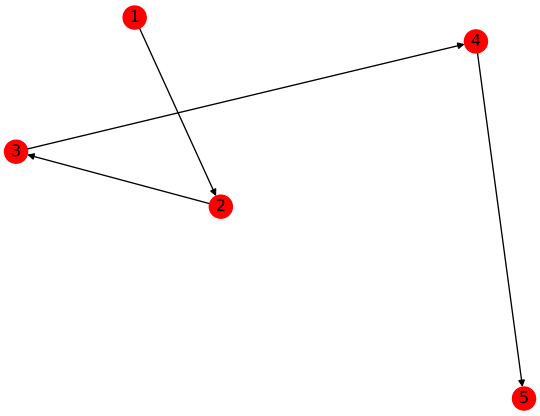

Figura 4: Grafo simple dirigido acíclico

5. Algoritmo Shell

Descripción Consiste en acomodar los nodos en circulos concentricos.

5.1. Código

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

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

nx.draw_shell(G, with_labels=True)

plt.show()
```

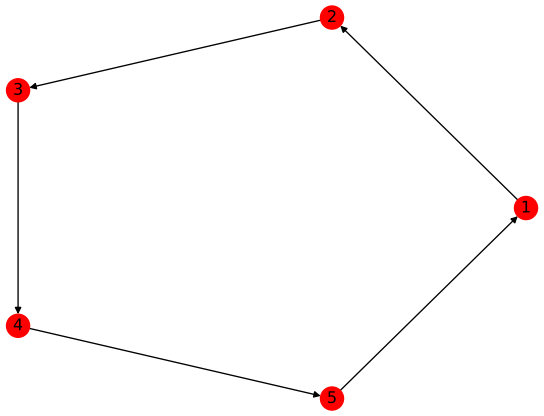

Figura 5: Gráfo simple dirigido cíclico

6. Algoritmo de acomodo de resorte

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

6.1. Código

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

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

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


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

plt.show()
```

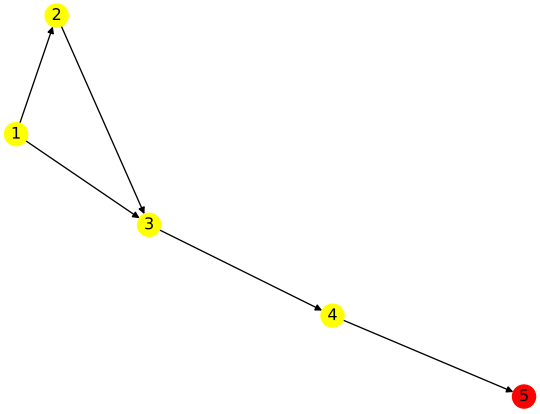

Figura 6: Gráfo simple dirigido reflexivo

7. Algortimo de acomodo de espectro

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

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

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

nx.draw_spectral(G, with_labels=True)

plt.show()
```

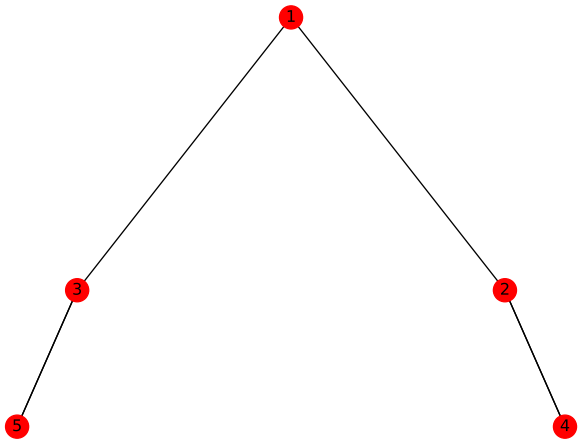

Figura 7: Multigrafo no dirigido acíclico

Referencias

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

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