Tarea No.2

5272

1. Circular Layout

Este algoritmo de diseño como su nombre lo indica puede ser útil cuando se desea estructurar la información con un patrón circular y se trata de disminuir los cruces entre los vértices [2]; es aplicado en redes sociales y administración de redes, con este se puede realizar estructuras de grupos y árbol dentro de una red [6].

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


G  = nx.DiGraph()


G.add_node("X")
G.add_nodes_from([1,2,3,4,5,6])

G.add_edges_from([("X",2),(2,4),(4,6),(6,5),(5,3),(3,1),(1,"X")])

nx.draw(G, pos=nx.circular_layout(G), with_labels= True)     # Dibuja el grafo G en forma
    circular
#plt.savefig("Tarea2_01.eps")
plt.show() #Se dibuja en pantalla
```

Tarea2_01.py

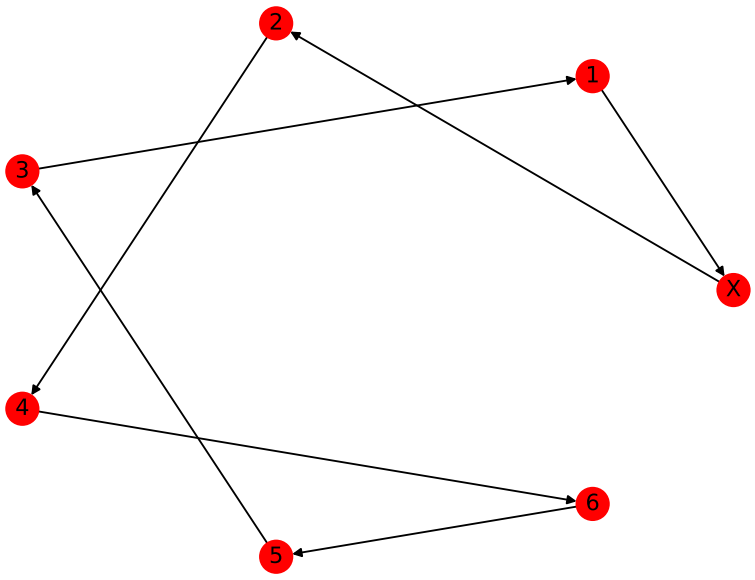

Figura 1: Circular Layout

2. Kamada Kawai Layout

Este algoritmo es para grafos conectados y no dirigidos; se relaciona con un sistema de resorte dinámico donde la fuerza entre dos vértices es inversamente proporcional al cuadrado de la distancia entre estos ,es decir, los vértices que tienen los resortes más fuertes se encuentran más cerca. [1].

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

G = nx.Graph()  #Se crea un grafo vacio

G.add_node(1) #Se crea el nodo raÃz
G.add_nodes_from([2,3,4])

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

color_map=[]
for node in G:
    if (node==2):
        color_map.append('blue')
    else:
        color_map.append('green')

nx.draw(G, pos=nx.kamada_kawai_layout(G), node_color=color_map, with_labels=True) #Se dibuja el
    grafo
plt.savefig("Tarea2_02.eps")
plt.show() #Se dibuja en pantalla
```

Tarea2_02.py

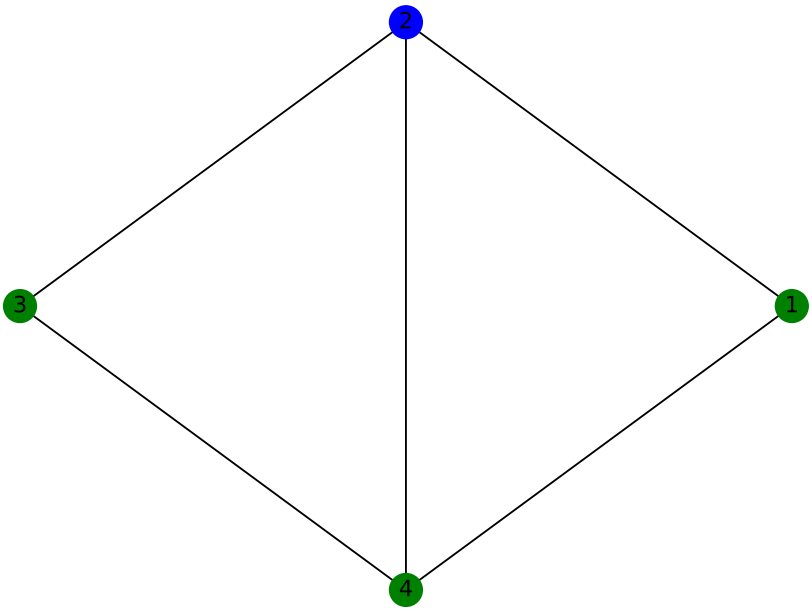

Figura 2: Kamada Kawai Layout

3. Random Layout

Este diseño se utiliza para cualquier tipo de grafo conectado y no conectado, planos y no planos.Se caracteriza por darle un orden aleatorio a los nodos de un grafo y tiene como limitación que para asegurarse de que los nodos no se superpongan con los margenes de la región del diseño,este algoritmo calcula las coordenadas al azar dentro de una región con las dimensiones más pequeñas qu la región del diseño [11].

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

G = nx.MultiGraph()  #Se crea un grafo vacio

G.add_node("A") #Se crea el nodo raÃz
G.add_nodes_from(["B","C","D","E"])

G.add_edge("B","C", color='red',weight=6)
G.add_edges_from([("A","B"),("B","C"),("C","D"),("D","E")],color='blue',weight=2)

edges = G.edges()

colors = []
weight = []

for (u,v,attrib_dict) in list(G.edges.data()):
    colors.append(attrib_dict['color'])
    weight.append(attrib_dict['weight'])

pos=nx.random_layout(G)

nx.draw(G,pos,edges=edges,edge_color=colors,width=weight,with_labels=True,font_size=8) #Se
    dibuja el grafo
plt.savefig("Tarea2_02.eps")
plt.show() #Se dibuja en pantalla
```

Tarea2_03.py

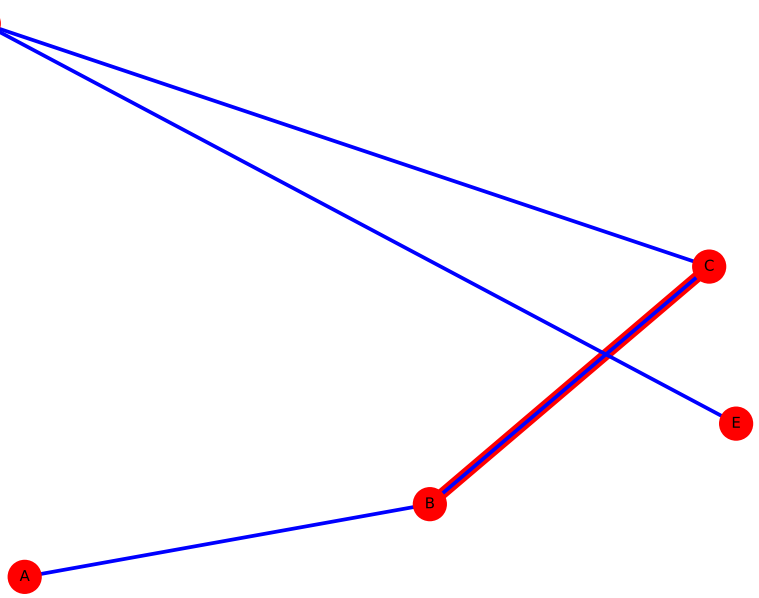

Figura 3: Random Layout

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

G = nx.MultiDiGraph()  #Se crea un grafo vacio

G.add_node("A") #Se crea el nodo raÃz
G.add_nodes_from(["C","B","R","G"])

G.add_edges_from([("A","C"),("C","A"),("C","B"),("B","C"),("B","R"),("R","B"),("R","G")])

nx.draw(G, pos=nx.random_layout(G), with_labels=True) #Se dibuja el grafo
plt.savefig("Tarea2_10.eps")
plt.show() #Se dibuja en pantalla
```

Tarea2_10.py

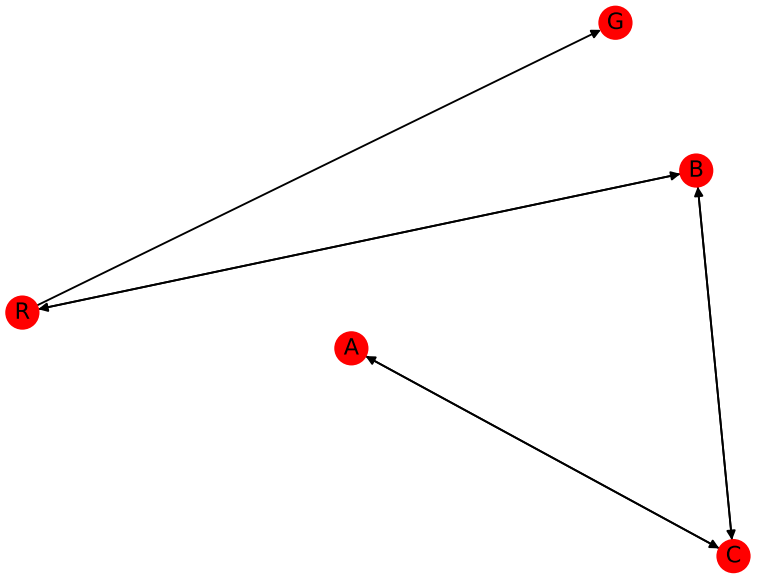

Figura 4: Random Layout

4. ForceAtlas2 Layout

En este diseño los nodos se rechazan entre sí,mientras que los bordes lo atraen como resortes,la posición de cada nodo depende de los otros nodos [9].

```
import networkx as nx
from fa2 import ForceAtlas2
import matplotlib.pyplot as plt

G = nx.Graph()

G.add_node(1)
G.add_nodes_from([2, 3, 4, 5, 6, 7, 8, 9, 10])

G.add_edges_from([(1, 10), (2, 10), (10, 3), (10, 4), (4, 5), (4, 6), (4, 7), (7, 4), (7, 8),
    (7, 9)])

#Este fragmento de codigo fue tomado de https://github.com/bhargavchippada/forceatlas2/blob/
    master/examples/forceatlas2-layout.ipynb

forceatlas2 = ForceAtlas2(

    outboundAttractionDistribution=True,
    linLogMode=False,
    adjustSizes=False,
    edgeWeightInfluence=1.0,

    jitterTolerance=1.0,
    barnesHutOptimize=True,
    barnesHutTheta=1.2,
    multiThreaded=False,

    scalingRatio=2.0,
    strongGravityMode=False,
    gravity=1.0,

    verbose=True)

positions = forceatlas2.forceatlas2_networkx_layout(G, pos=None, iterations=2000)
nx.draw_networkx_nodes(G, positions, node_size=30, with_labels=False, node_color="green",
    alpha=0.7)
nx.draw_networkx_edges(G, positions, edge_color="blue", alpha=0.06)
plt.axis('off')
plt.savefig("Tarea2_10.eps")
plt.show()
```

Tarea2_4.py

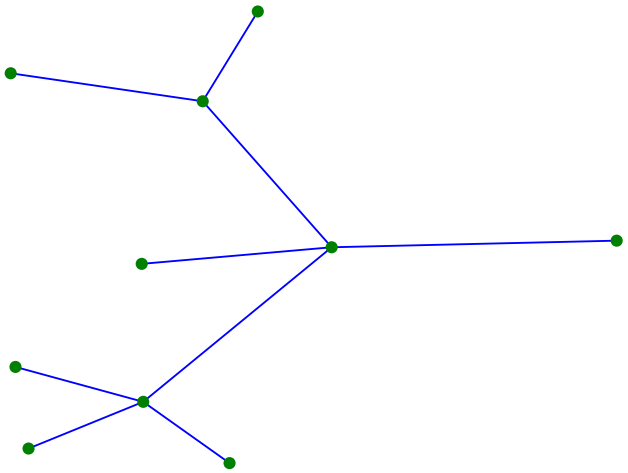

Figura 5: ForceAtlas2 Layout

5. Shell Layout

Este diseño posiciona los nodos en círculos concéntricos [10], basandose en la distancia entre el nodo central al resto de los nodos,por lo que los nodos se distribuyen de forma circular;este tiene como ventaja que su diseño hace que sea un grafo visualmente de fácil entender [4]

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

G = nx.Graph()  #Se crea un grafo vacio

G.add_node(1)
G.add_nodes_from([2,3,4,5,6])
G.add_edges_from([(1,2),(1,3),(1,4),(1,5),(1,6)])
G.add_edges_from  ([(2,3)])
G.add_edges_from  ([(3,4)])
G.add_edges_from  ([(4,5)])
G.add_edges_from  ([(5,6)])
G.add_edges_from  ([(6,2)])

shells=[[1],[2,3,4,5,6]]

nx.draw(G,pos=nx.shell_layout(G,shells),with_labels=True) #Se dibuja el grafo
plt.savefig("Tarea2_05.eps")
plt.show() #Se dibuja en pantalla
```

Tarea2_05.py

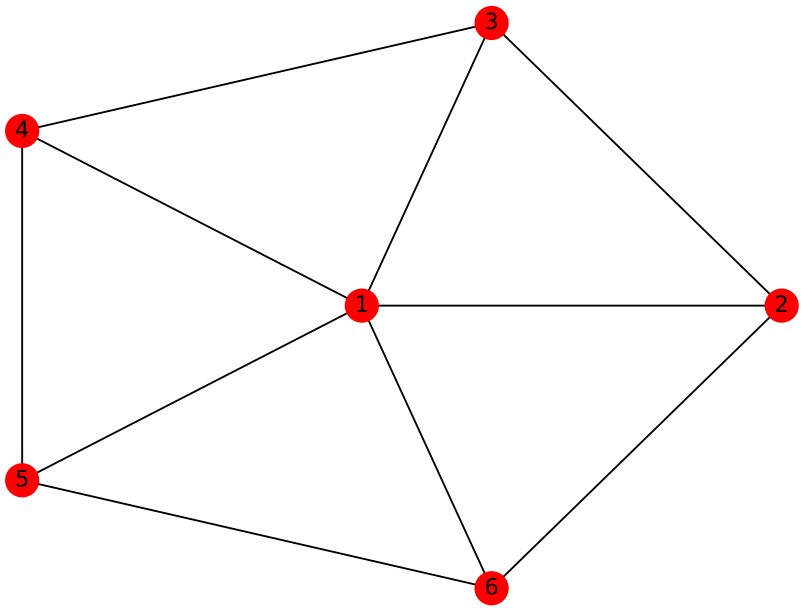

Figura 6: Shell Layout

6. Kamada kawai Layout

Este algoritmo es dirigido por la fuerza entre dos nodos cualesquiera,donde los mismo se representan por anillos de acero y las aristas son los resortes entre ellos.Las fuerzas de atracción y de repulsión son análogas a la fuerza de resorte,donde la suma de las fuerzas determinan en que dirección se debe mover un nodo [5]; es utilizado en grafos no dirigidos muy grandes y garantiza que los nodos cercanos sean ubicados en la misma vencidad y los lejanos se ubican uno lejos de otros.

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

G = nx.MultiGraph()  #Se crea un grafo vacio

G.add_node(1) #Se crea el nodo raÃz

G.add_nodes_from([2,3,4,5,6,7,8,9])
G.add_edge(1,2,color='blue',weight=4)
G.add_edge(3,2,color='blue',weight=4)
G.add_edge(3,4,color='blue',weight=4)
G.add_edge(3,5,color='blue',weight=4)
G.add_edge(6,5,color='blue',weight=4)
G.add_edge(6,7,color='blue',weight=4)
G.add_edge(8,7,color='blue',weight=4)
G.add_edge(8,7,color='blue',weight=4)
G.add_edge(9,7,color='blue',weight=4)

G.add_edges_from([(2,1),(2,3),(4,3),(5,3),(5,6),(7,6),(7,8),(9,7)],color='red',weight=1)

edges= G.edges()

colors=[]
weight=[]

for (u,v,attrib_dict) in list(G.edges.data()):
    colors.append(attrib_dict['color'])
    weight.append(attrib_dict['weight'])

pos=nx.fruchterman_reingold_layout(G)


nx.draw(G,pos,edges=edges,edge_color=colors,width=weight,with_labels=True,font_size=8) #Se
    dibuja el grafo
plt.savefig("Tarea2_06.eps")
plt.show() #Se dibuja en pantalla
```

Tarea2_06.py

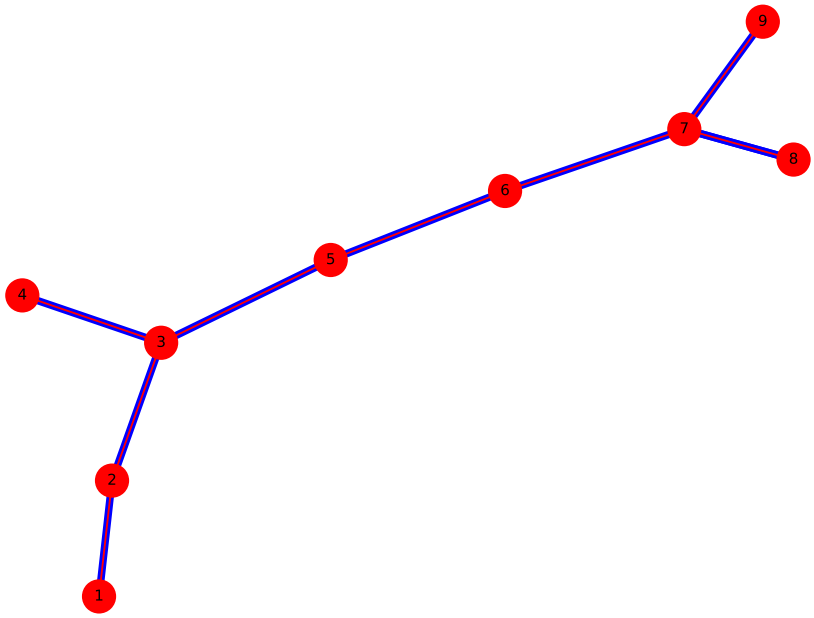

Figura 7: Kamada kawai Layout

7. Spring Layout

En este diseño los nodos serían los cuerpos y los bordes los resortes que conectan a estos, los mismo son los que proporcionan fuerzas entre los nodos; estos se mueven de acuerdo con las fuerzas que se ejercen hasta alcanzar la mínima energía local [7].

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

G = nx.Graph()  #Se crea un grafo vacio

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

nx.draw(G,pos=nx.spring_layout(G),with_labels=True) #Se dibuja el grafo
plt.savefig("Tarea2_07.eps")
plt.show() #Se dibuja en pantalla
```

Tarea2_07.py

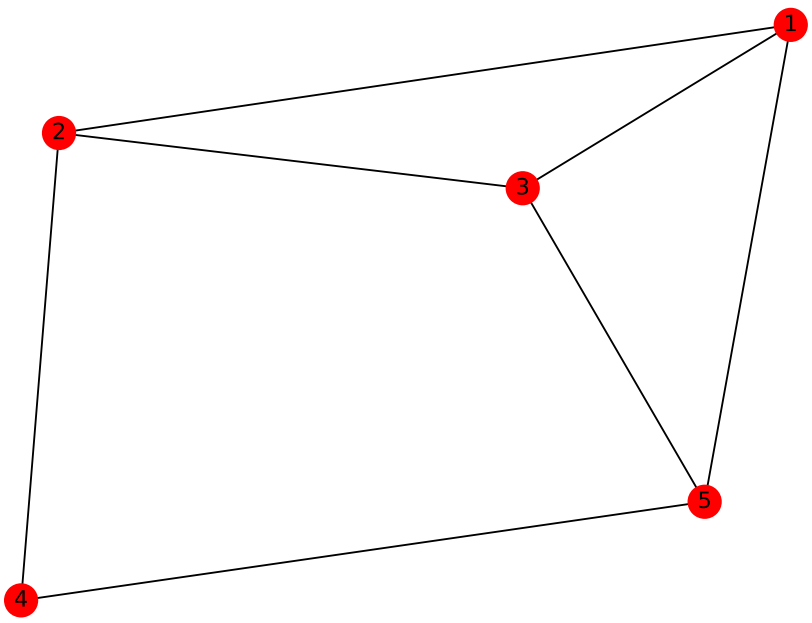

Figura 8: Spring Layout

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

G =nx.MultiDiGraph()

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

nx.draw(G, pos=nx.spring_layout(G), with_labels=True) #Se dibuja el grafo
plt.savefig("Tarea2_11.eps")
plt.show() #Se dibuja en pantalla
```

Tarea2_11.py

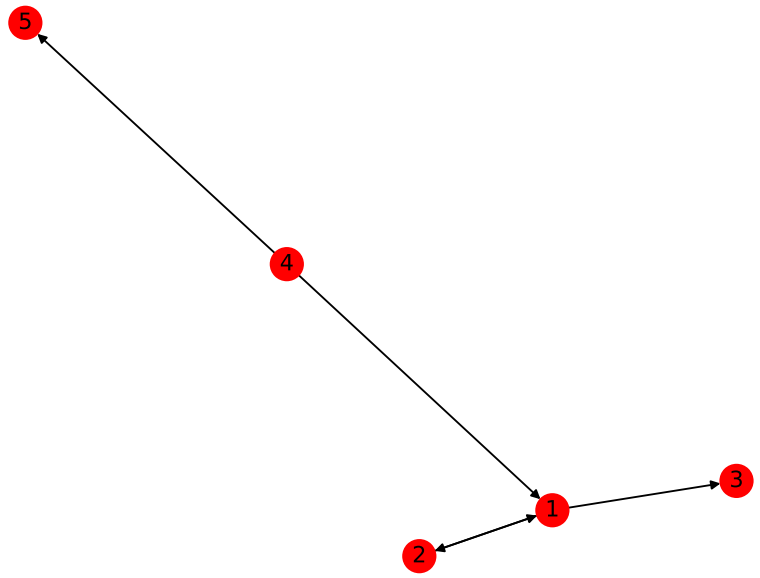

Figura 9: Spring Layout

8. Spectral Layout

Este diseño utiliza los vectores de matrices relacionadas con gráficas como las de adyacencia, tiene como ventaja la capacidad de calcular diseños óptimos y presentar un tiempo de calculo muy rápido [3].

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

G = nx.MultiDiGraph()  #Se crea un grafo vacio

G.add_node(1) #Se crea el nodo raÃz
G.add_nodes_from([2,3,4,5])
G.add_edge(2,1,color='red',weight=1)
G.add_edges_from([(1,2),(2,3),(3,4),(4,5),(5,2)],color='blue',weight=3)

edges= G.edges()

colors=[]
weight=[]

for (u,v,attrib_dict) in list(G.edges.data()):
    colors.append(attrib_dict['color'])
    weight.append(attrib_dict['weight'])

nx.draw(G,pos=nx.spectral_layout(G), edges=edges,edge_color=colors,width=weight,with_labels=
    True,font_size=8) #Se dibuja el grafo
plt.savefig("Tarea2_07.eps")
plt.show() #Se dibuja en pantalla
```

Tarea2_08.py

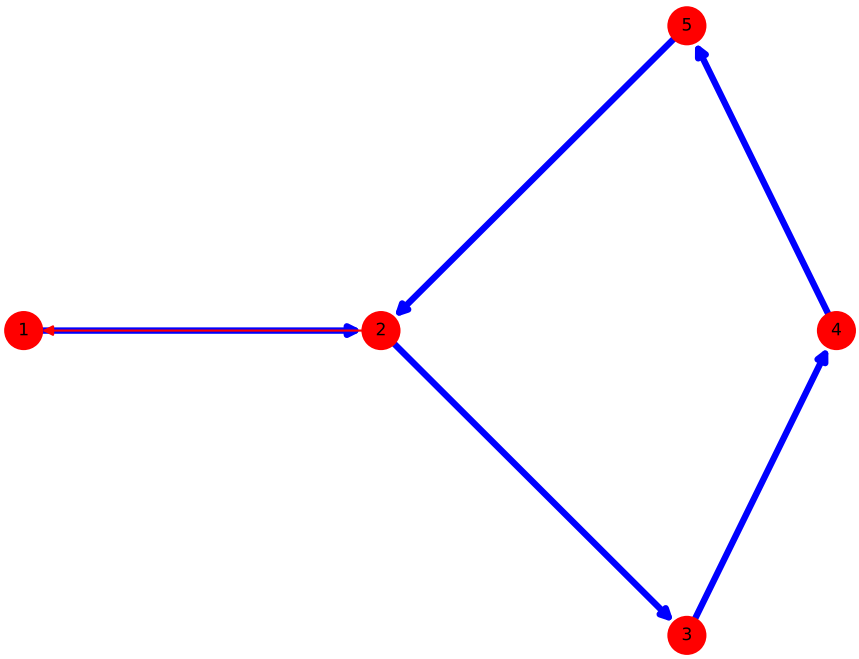

Figura 10: Spectral Layout

9. Bipartite Layout

Estos grafos representan las relaciones entre dos conjuntos y permiten la exploración paso a paso a gran escala.Los nodos de un mismo conjunto no se encuentran conectados entre sí [8].

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

G = nx.DiGraph()  #Se crea un grafo vacio

G.add_nodes_from(['s','x'],bipartite=0)
G.add_nodes_from(['w','z','y','t'],bipartite=1)
G.add_edges_from([('s','w'),('s','z'),('s','t'),('x','y'),('z','x')])


nx.draw(G,pos=nx.bipartite_layout(G,['s','x']),with_labels=True) #Se dibuja el grafo
plt.savefig("Tarea2_06.eps")
plt.show() #Se dibuja en pantalla
```

Tarea2_09.py

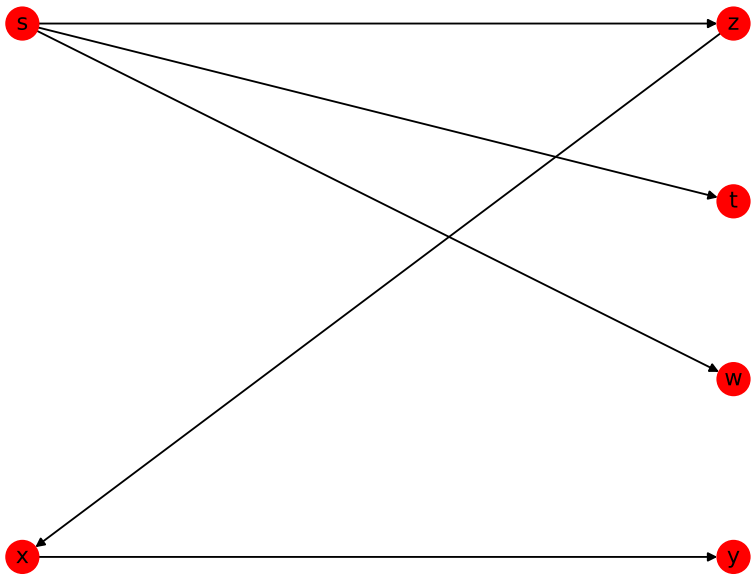

Figura 11: Bipartite Layout

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

G = nx.Graph()  #Se crea un grafo vacio

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

nx.draw(G, pos=nx.bipartite_layout(G,[1,2]),with_labels=True) #Se dibuja el grafo
plt.savefig("Tarea2_12.eps")
plt.show() #Se dibuja en pantalla
```

Tarea2_12.py

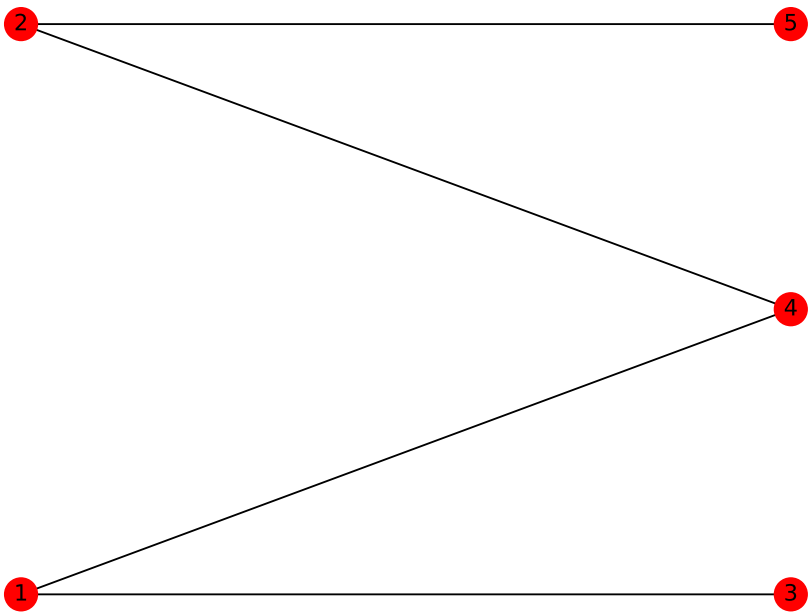

Figura 12: Bipartite Layout

Referencias

[1] Un algoritmo para dibujar gráficos generales no dirigidos. *Cartas de procesamiento de información.*

[2] Mehmet Esat Belviranlı. A circular layout algorithm for clustered graphs. Master's thesis, bilkent university, 2009.

[3] Ulrik Brandes, Daniel Fleischer, and Thomas Puppe. Dynamic spectral layout with an application to small worlds. *Journal of Graph Algorithms and Applications*, 11(2):325–343, 2007.

[4] Ken Cherven. *Mastering Gephi network visualization.* Packt Publishing Ltd, 2015.

[5] Thomas MJ Fruchterman and Edward M Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.

[6] Emden R Gansner and Yehuda Koren. Improved circular layouts. In *International Symposium on Graph Drawing*, pages 386–398. Springer, 2006.

[7] Gerardo Huck. Posicionamiento automático de etiquetas en grafos. B.S. thesis, Facultad de Ciencias Exactas, Ingeniería y Agrimensura. Universidad Nacional, 2014.

[8] Takao Ito, Kazuo Misue, and Jiro Tanaka. Drawing clustered bipartite graphs in multi-circular style. In *2010 14th International Conference Information Visualisation*, pages 23–28. IEEE, 2010.

[9] Mathieu Jacomy, Tommaso Venturini, Sebastien Heymann, and Mathieu Bastian. Forceatlas2, a continuous graph layout algorithm for handy network visualization designed for the gephi software. *PLOS ONE*, 9(6): 1–12, 06 2014. URL `https://doi.org/10.1371/journal.pone.0098679`.

[10] NetworkX. Source code for networkx.drawing.layout, 2018. `https://networkx.github.io/documentation/latest/_modules/networkx/drawing/layout.html`, Last accessed on 2019-02-13.

[11] Rogue Wave Software. Random layout, 2016. `https://docs.roguewave.com/visualization/views/6.1/views.html#page/Options%2Flayouts.51.19.html%23`, Last accessed on 2019-02-23.