Tarea 2 Optimización de Flujo en Redes

Orlando Lázaro Ruiloba Torres 5270

26 de febrero de 2019

1. Algoritmo de acomodo bipartito

El diseño de este algoritmo se crea colocando primero los vértices en dos filas, de acuerdo con sus tipos. Luego, las posiciones dentro de las filas se optimizan para minimizar los cruces de arcos, utilizando el algoritmo de Sugiyama [1]. Este algoritmo de acomodo actualmente solo funciona en dos dimensiones [8].

El código en python se muestra a continuación:

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

GDA = nx.DiGraph()

GDA.add_nodes_from(["E1","E2","E3"], bipartite=0)
GDA.add_nodes_from(["E4","E5","E6","E7"], bipartite=1)

GDA.add_edges_from([("E1","E4"),("E1","E5"),("E2","E6"),("E2","E7"),("E3","E4"),("E3","E7")])

nx.draw(GDA, pos=nx.bipartite_layout(GDA,["E1","E2","E3"]), node_size = 2000, node_color = 'y',
       node_shape = 'o', with_labels=True)
plt.draw()
plt.savefig("GDA.eps")
plt.show()
```

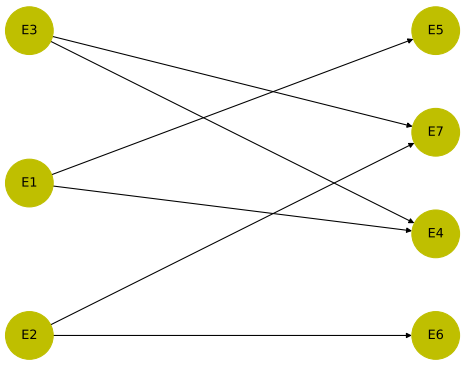

Figura 1: Algoritmo bipartito en grafo dirigido acíclico

2. Algoritmo de acomodo circular

Es un algoritmo de diseño que permite colocar los vértices en topologías de anillo y estrella interconectados. Produce diseños que enfatizan las estructuras de grupo y árbol dentro de una red. Crea particiones de nodo mediante el análisis de la estructura de conectividad de la red, y organiza las particiones como círculos separados. Los círculos en sí están dispuestos en forma de árbol radial [2].

El código en python es:

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

MNDC = nx.MultiGraph()
MNDC.add_nodes_from(["1","9"])

MNDC.add_edges_from([("1","2")], color='lightblue', weight=8)
MNDC.add_edges_from([("2","3"),("3","4"),("4","5"),("5","6"),("6","7"),("7","8"),("8","9"),
                     ("2","1"),("1","6"),("2","7"),("4","9")], color='black', weight=2)

edges = MNDC.edges()

colors = []
weight = []

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

color_map = []
for node in MNDC:
    if (node == "1" or node == "2"):
        color_map.append('blue')
    else:
        color_map.append('red')

nx.draw_circular(MNDC, edges=edges, edge_color=colors, width=weight, node_size = 1000,
                 node_color = color_map, node_shape = 'o', with_labels=True)
plt.draw()
plt.savefig("MNDC.eps")
plt.show()
```

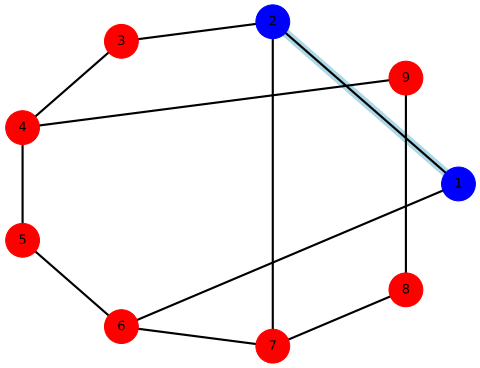

Figura 2: Algoritmo circular en multigrafo no dirigido cíclico

3. Algoritmo de acomodo kamada kawai

Este algoritmo de acomodo se basa en la idea de usar solo fuerzas de resorte entre todos los pares de vértices, el criterio estético que considera es que la distancia entre los vértices debe ser igual a la distancia teórica. Mantiene implícitamente los dos criterios anteriores [11].

El código en python se muestra a continuación:

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

GNDR = nx.Graph()

GNDR.add_nodes_from(["C1","C7"])
GNDR.add_edges_from([("C1","C2"),("C2","C3"),("C3","C4"),("C4","C5"),("C5","C3"),("C3","C6"),
                     ("C6","C7"),("C7","C7")])

color_map = []
for node in GNDR:
    if (node == "C7"):
        color_map.append('blue')
    else:
        color_map.append('red')

layout = nx.kamada_kawai_layout(GNDR)

nx.draw(GNDR, pos=layout, node_size = 1000, node_color = color_map, node_shape = 'o',
    with_labels=True)
plt.draw()
plt.savefig("GNDR.eps")
plt.show()
```

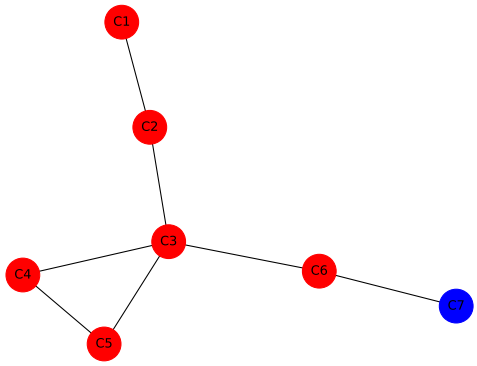

Figura 3: Algoritmo kamada kawai en grafo simple no dirigido reflexivo

4. Algoritmo de acomodo aleatorio

Es un algoritmo que posiciona los vértices uniformemente al azar en el cuadrado unitario. Para cada vértice se genera una posición, al elegir cada una de las coordenadas tenues de la forma mencionada anteriormente en el intervalo [0.0, 1.0] [6].

El código en pyhton es el siguiente:

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

GNDC = nx.Graph()

GNDC.add_node("F")
GNDC.add_nodes_from(["C1","C2","C3","C4"])

GNDC.add_edges_from([("F","C3"),("C3","C4"),("C4","C2"),("C2","C1"),("C1","F")])

nx.draw_random(GNDC, node_size = 2000, node_color = 'y', node_shape = 'o', with_labels=True)
plt.draw()
plt.savefig("GNDC.eps")
plt.show()
```

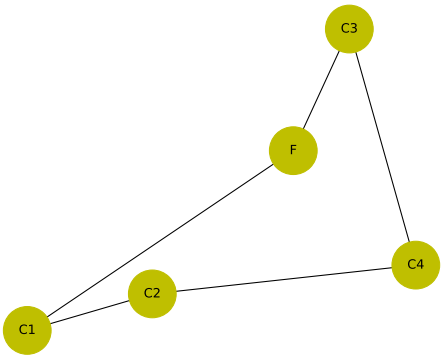

Figura 4: Algoritmo aleatorio en grafo no dirigido cíclico

5. Algoritmo de acomodo espectral

El código en python se muestra a continuación:

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

MNDR = nx.MultiGraph()

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

color_map = []
for node in MNDR:
    if (node == "5"):
        color_map.append('yellow')
    else:
        color_map.append('red')

nx.draw_spectral(MNDR, node_size = 1000, node_color = color_map, node_shape = 'o', with_labels=True)
plt.draw()
plt.savefig("MNDR.eps")
plt.show()
```

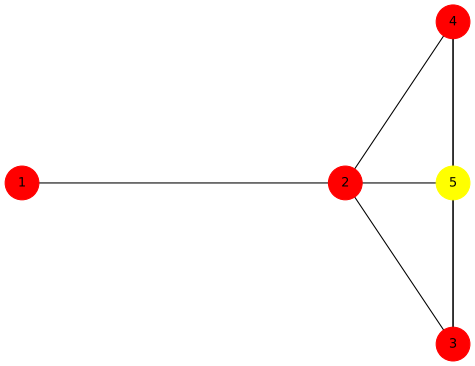

Figura 5: Algoritmo espectral en multigrafo no dirigido reflexivo

6. Algoritmo de acomodo shell

Este algoritmo de acomodo es interesante, ya que le permite al diseñador seleccionar subgrupos de nodos para posicionarlos en círculos concéntricos, en este sentido es menos automático que otros diseños, pero tiene la ventaja crítica de que se puede usar para resaltar características específicas de la red [3].

El código en python es:

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

GDR = nx.DiGraph()

GDR.add_nodes_from(["C1","C5"])
GDR.add_edges_from([("C1","C2"),("C2","C3"),("C3","C4"),("C4","C2"),("C5","C1"),("C5","C5")])

color_map = []
for node in GDR:
    if (node == "C5"):
        color_map.append('blue')
    else:
        color_map.append('red')

nx.draw_shell(GDR, node_size = 2000, node_color = color_map, node_shape = 'o', with_labels=True)
plt.draw()
plt.savefig("GDR.eps")
plt.show()
```

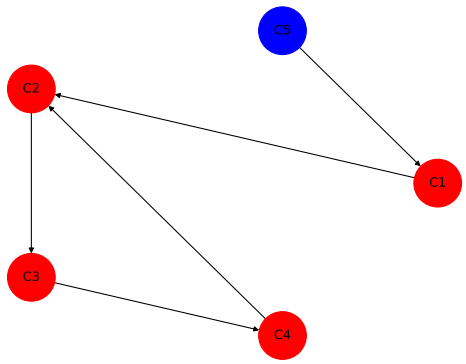

Figura 6: Algoritmo shell en grafo dirigido reflexivo

7. Algoritmo de acomodo resorte

En este caso el algoritmo posiciona los nodos basándose en el algoritmo de fuerza dirigida de Fruchterman-Reingold [7]. Cada elemento se trata como una partícula con una carga eléctrica similar que repele otros elementos. Los conectores actúan como resortes que vuelven a unir los elementos conectados. El algoritmo es bueno para resaltar grupos de objetos relacionados e identificar simetría en el grafo [10].

El código en python se muestra a continuación:

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

GNDA = nx.Graph()

GNDA.add_node("Gerente")
GNDA.add_nodes_from(["Subgerente","Asistente","Coordinador","Planificador","Organizador","Supervisor"])

GNDA.add_edges_from([("Gerente","Subgerente"),("Subgerente","Asistente")])
GNDA.add_edges_from([("Asistente","Coordinador"),("Asistente","Planificador")])
GNDA.add_edges_from([("Asistente","Organizador"),("Asistente","Supervisor")])

nx.draw_spring(GNDA, node_size = 5500, node_color = 'y', node_shape = 's', with_labels=True)
plt.draw()
plt.savefig("GNDA.eps")
plt.show()
```

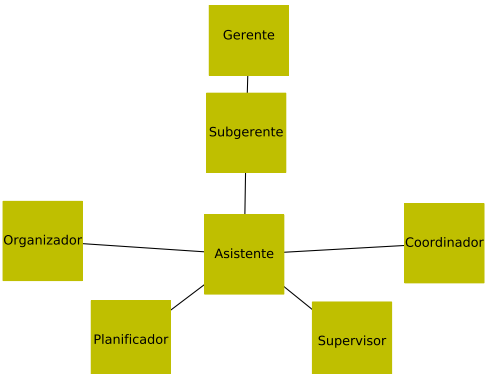

Figura 7: Algoritmo resorte en grafo no dirigido acíclico

8. Algoritmo de acomodo espectral

Este algoritmo posiciona los nodos utilizando los vectores propios del grafo Laplaciano, el diseño muestra posibles agrupaciones de nodos que son una aproximación de la relación de corte [9].

El código en python es el siguiente:

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

MDC = nx.MultiDiGraph()

MDC.add_nodes_from(["E1","E5"])
MDC.add_edges_from([("E1","E2"),("E2","E1"),("E2","E3"),("E3","E2"),("E3","E4"),("E4","E3"),
                    ("E4","E5"),("E5","E4")])

nx.draw_spectral(MDC, node_size = 1000, node_color = 'yellow', node_shape = 'o', with_labels=True)
plt.draw()
plt.savefig("MDC.eps")
plt.show()
```

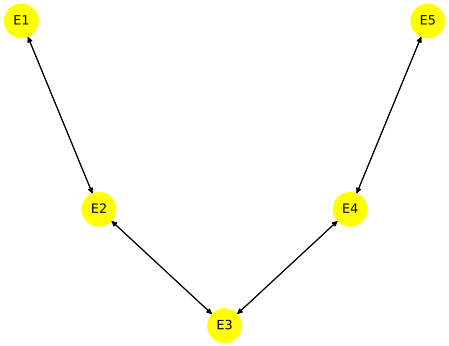

Figura 8: Algoritmo espectral en multigrafo dirigido cíclico

9. Algoritmo de acomodo Fruchterman-Reingold

La idea de este algoritmo de acomodo es considerar una fuerza entre dos nodos cualesquiera, en este algoritmo los nodos están representados por anillos de acero y los arcos son resortes entre ellos. La idea básica es minimizar la energía del sistema moviendo los nodos y cambiando las fuerzas entre ellos. En este algoritmo, la suma de los vectores de fuerza determina en que dirección se debe mover un nodo [4].

El código en python es:

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

MDA = nx.MultiDiGraph()

MDA.add_nodes_from(["C1","C5"])
MDA.add_edges_from([("C3","C4")], color='lightblue', weight=4)
MDA.add_edges_from([("C1","C3"),("C2","C3"),("C3","C4"),("C3","C4"),("C4","C5")], color='black',
    weight=1)

edges = MDA.edges()

colors = []
weight = []

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

color_map = []
for node in MDA:
    if (node == "C3" or node == "C4"):
        color_map.append('yellow')
    else:
        color_map.append('blue')

frl = nx.fruchterman_reingold_layout(MDA, k=0.40, iterations=40)

nx.draw(MDA, pos=frl, edges=edges, edge_color=colors, width=weight, node_size = 1000,
        node_color = color_map, node_shape = 'o', with_labels=True)
plt.draw()
plt.savefig("MDA.eps")
plt.show()
```

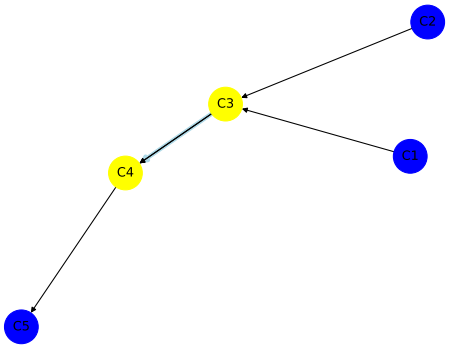

Figura 9: Algoritmo fruchterman-reingold en multigrafo dirigido acíclico

10. Algoritmo de acomodo ForceAtlas2

Es un algoritmo contínuo que permite manipular el grafo mientras se está renderizando. Cuenta con una velocidad de convergencia adaptativa única que permite que la mayoría de los grafos converjan de manera más eficiente. Propone ajustes resumidos, enfocados en qué impacto tiene la forma del grafo (escalado, gravedad,...) [5].

El código en python es el siguiente:

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

MNDA = nx.MultiGraph()

MNDA.add_nodes_from(["S1", "S5"])
MNDA.add_edges_from([("S3", "S4"), ("S4", "S5")], color='blue', weight=5)
MNDA.add_edges_from([("S1", "S2"), ("S2", "S3"), ("S2", "S4"), ("S2", "S5"), ("S3", "S4"),
                     ("S4", "S5"),("S3", "S5")], color='black', weight=2)

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

#Tomado de: https://github.com/bhargavchippada/forceatlas2/blob/master/README.md
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(MNDA, pos=None, iterations=1000)
nx.draw_networkx_nodes(MNDA, positions, node_size=500, with_labels=False, node_color="blue",
    alpha=0.4)
nx.draw_networkx_edges(MNDA, positions, edge_color="black", alpha=0.05)
plt.axis('off')
plt.savefig("MNDA.eps")
plt.show()
```

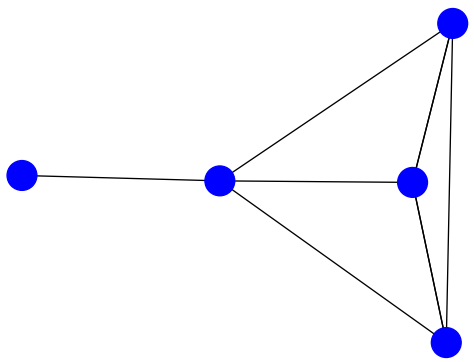

Figura 10: Algoritmo ForceAtlas2 en multigrafo no dirigido acíclico

11. Algoritmo de acomodo resorte

El código en python se muestra a continuación:

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

GDC = nx.DiGraph()

GDC.add_nodes_from(["S1","S5"])
GDC.add_edges_from([("S1","S2"),("S2","S5"),("S3","S4"),("S4","S5"),("S5","S1")])

nx.draw_spring(GDC, node_size = 1000, node_color = 'y', node_shape = 'o', with_labels=True)
plt.draw()
plt.savefig("GDC.eps")
plt.show()
```

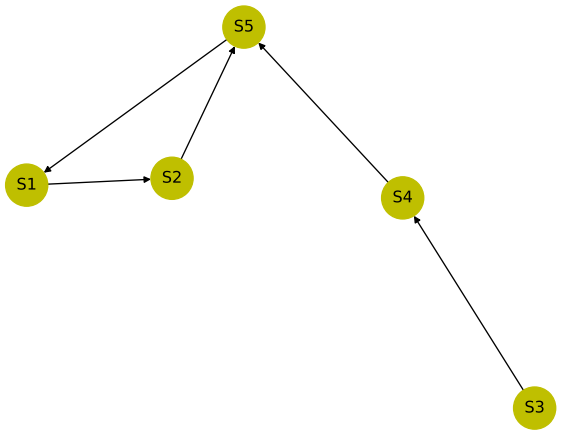

Figura 11: Algoritmo resorte en grafo dirigido cíclico

12. Algoritmo de acomodo circular

El código en python es:

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

MDR = nx.MultiDiGraph()

MDR.add_nodes_from(["D1","D5"])
MDR.add_edges_from([("D1","D2"),("D1","D4"),("D2","D3"),("D3","D2"),("D3","D4"),("D3","D5"),("D4","D5"),
                    ("D5","D5")])

color_map = []
for node in MDR:
    if (node == "D5"):
        color_map.append('yellow')
    else:
        color_map.append('red')

nx.draw_circular(MDR, node_size = 1000, node_color = color_map, node_shape = 'o', with_labels=True)
plt.draw()
plt.savefig("MDR.eps")
plt.show()
```

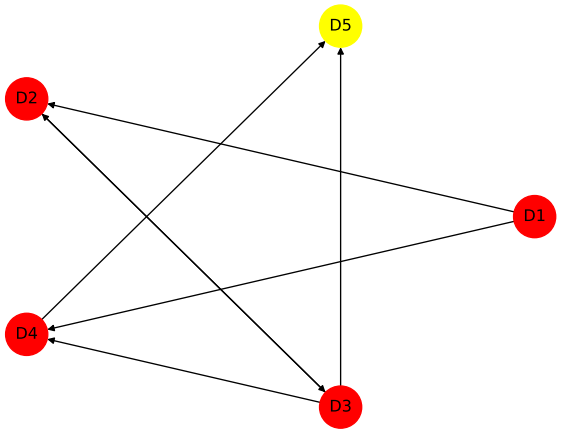

Figura 12: Algoritmo circular en multigrafo dirigido reflexivo

Referencias

[1] Gabor Csardi. Layout as bipartite: Simple two-row layout for bipartite graphs. Last access in 02-17-2019 to urlhttp://rdrr.io/cran/igraph/man/layout as bipartite.htmlheading-5, 2019.

[2] The diagramming company. Developers guide: Analysis and layout. Last access in 02-17-2019 to urlhttp://docs.yworks.com/yfilesflex/doc/dguide-layout/circular layouter.html, 2015.

[3] Simon Dobson. Concepts: networks and geometry. Last access in 02-23-2019 to urlhttp://simondobson.org/2017/04/21/concepts-networks-and-geometry/, 2017.

[4] Sébastien Heymann. Fruchterman reingold. Last access in 02-23-2019 to urlhttp://github.com/gephi/gephi/wiki/Fruchterman-Reingold, 2015.

[5] Mathieu Jacomy. Forceatlas2, the new version of our home-brew layout. Last access in 02-25-2019 to urlhttp://gephi.wordpress.com/2011/06/06/forceatlas2-the-new-version-of-our-home-brew-layout/, 2011.

[6] Networkx. Random layout. Last access in 02-21-2019 to urlhttp://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.drawing.layout.randomlayout.html, 2015.

[7] Networkx. Networkx drawing layout spring layout. Last access in 02-23-2019 to urlhttp://networkx.github.io/documentation/stable/reference/generated/networkx.drawing.layout.spring layout.html, 2018.

[8] Networkx. Networkx drawing layout bipartite layout. Last access in 02-17-2019 to urlhttp://networkx.github.io/documentation/latest/reference/generated/networkx.drawing.layout.bipartite layout.html, 2019.

[9] Networkx. Spectral layout. Last access in 02-23-2019 to urlhttp://networkx.github.io/documentation/latest/reference/generated/networkx.drawing.layout.spectral layout.html, 2019.

[10] Sparx Systems. Spring layout. Last access in 02-23-2019 to urlhttp://sparxsystems.com/enterprisearchitectuserguide/14.0/modelingtools/spring layout.html, 2019.

[11] Andrés Aiello y Rodrigo I. Silveira. Trazado de grafos mediante métodos dirigidos por fuerza. Last access in 02-21-2019 to urlhttp://slideplayer.es/slide/2343740/, 2018.