Tarea 4

Jesus Angel Patlán Castillo (5261)

2 de abril de 2019

En esta tarea se analizan la complejidad que se tiene en la utilización de métodos algorítmicos de flujo máximo en distintos tipos de grafos, los cuales son generados por algoritmos de generación de grafos proporcionados por librerías de Python. Los algoritmos se obtuvieron por medio de la librería NetworkX [3] de Python [1], la librería Matplotlib [4] es utilizada para generar las gráfica de correlación entre los efectos estudiados. El código empleado se obtuvo consultando la documentación oficial de la librería NetworkX [2]. Las imágenes y el código se encuentran disponibles directamente en mi repositorio [5].

1. Metodología

La librería NetworkX tiene a la mano diversos algoritmos para resolver distintos algoritmos para poder resolver problemas de flujo máximo. Particularmente para esta investigación, se realiza el análisis de los algoritmos:

- Flujo Máximo: El algoritmo de flujo máximo nos permite encontrar el flujo máximo que se puede dar entre dos nodos del grafo (llamados fuente y sumidero). En la librería de NetworkX esta disponible por medio de la función `maximum flow(G,N1,N2)`, donde el parámetro G es el grafo, N1 el nodo fuente y N2 el nodo sumidero:

```
nx.maximum_flow(G1,G1S,G1T)
```

- Valor de flujo máximo: El algoritmo de valor de flujo máximo se utiliza para encontrar el valor máximo de flujo entre un par de nodos. En la librería de NetworkX esta disponible por medio de la función `maximum flow value(G,N1,N2)`, donde el parámetro G es el grafo, $N1$ es el nodo fuente y $N2$ es el nodo sumidero:

```
nx.maximum_flow_value(G1, G1S, G1T)
```

- Valor de corte mínimo: El algoritmo valor de corte mínimo nos permite encontrar el valor mínimo entre el corte (una partición de nodos del grafo) que permita minimizar los pesos entre los grafos. En la librería de NetworkX esta disponible por medio de la función `minimum cut value(G,N1,N2)`, donde el parámetro G es el grafo, $N1$ es el nodo fuente y $N2$ es el nodo sumidero:

```
nx.minimum_cut_value(G1, G1S, G1T)
```

Cada uno de estos algoritmos fueron ejecutados sobre diez grafos diferentes generados por tres métodos de generación de grafos distintos, los cuales son:

- Grafo paleta: Un grafo paleta es un grafo que consiste en la unión entre un grafo completo y un "puente"sobre uno de los nodos del grafo. En la librería de NetworkX esta disponible por medio de la función `lollipop graph(N1,N2)`, donde el parámetro $N1$ es la dimensión del grafo completo y $N2$ es la dimensión del "puente":

```
G1 = nx.lollipop_graph(log, 2)
```

- Grafo Turan: El grafo turan es un grafo completo multipartito de que contiene n nodos con r subconjuntos disjuntos. En la librería de NetworkX esta disponible por medio de la función `turan graph(n,r)`, donde el parámetro n es la cantidad de nodos del grafo completo y r es la cantidad de subconjuntos disjuntos:

```
G2 = nx.turan_graph(log, 2)
```

- Grafo escalera: El grafo escalera es un grafo que contiene dos caminos de n nodos, sobre el cual cada par esta conectado por una única arista. En la librería de NetworkX esta disponible por medio de la función `ladder graph(n)`, donde el parámetro n es la dimensión del grafo:

```
G3 = nx.ladder_graph(log)
```

Para cada combinación de métodos de flujo máximo y de generación de grafo, se realizaron 10 replicas de grafos distintos, para cada uno de los cuales se realizaron 5 repeticiones distintas para 5 pares de fuente-sumidero distintos. Además, se consideraron 4 distintos ordenes de tamaño para cada grafo, considerando una escala logarítmica en base 2^n con valores de $n \in \{7, 8, 9, 10\}$. Después de realizar las replicas, se obtuvo el tiempo que tomo en ejecutar cada algoritmo en cada uno de los grafos, y se realizaron las pruebas estadísticas ANOVA para determinar si el tiempo de ejecución es afectado por el orden del grafo, el algoritmo de flujo máximo utilizado, el algoritmo generador del grafo y la densidad de cada grafo; y si existe una correlación entre estos factores. A cada arista de cada grafo se le asigno un peso dado por una distribución normal con media 10 y desviación 2.5. Las siguientes líneas de código representan la generación de la recopilación de datos de las replicas, la matriz de correlación y la tabla de ANOVA:

```
#Orden logaritmico
for logarithmOrder in range(3,5):
    print('Orden: ',logarithmOrder)
    log = 2**logarithmOrder
    ban=True
    # 3 Metodos de generaci n distintos
```

```
    G1 = nx.lollipop_graph(log, 2)
    for e in G1.edges():
        G1[e[0]][e[1]]['capacity'] = np.random.normal(mu, sigma)
    G2 = nx.turan_graph(log, 2)
    for e in G2.edges():
        G2[e[0]][e[1]]['capacity'] = np.random.normal(mu, sigma)
    G3 = nx.ladder_graph(log)
    for e in G3.edges():
        G3[e[0]][e[1]]['capacity'] = np.random.normal(mu, sigma)
    for newPair in range(5):
        print('Pareja: ',newPair)
        #10 Grafos
        for graphRepetition in range(10):
            if ban:
                ban=False
                while G1T==G1S:
                    G1S = choice(list(G1.nodes()))
                    G1T = choice(list(G1.nodes()))
                while G2T==G2S:
                    G2S = choice(list(G2.nodes()))
                    G2T = choice(list(G2.nodes()))
                while G3T==G3S:
                    G3S = choice(list(G3.nodes()))
                    G3T = choice(list(G3.nodes()))
            # GRAFO 1
            g1_start_time = time.time()
            nx.maximum_flow(G1,G1S,G1T)
            g1_end_time = time.time() - g1_start_time
            totalTests.append(Test(0, 0, logarithmOrder, nx.density
    (G1), g1_end_time))

            g1_start_time = time.time()
            nx.maximum_flow_value(G1, G1S, G1T)
            g1_end_time = time.time() - g1_start_time
            totalTests.append(Test(0, 1, logarithmOrder, nx.density
    (G1), g1_end_time))

            g1_start_time = time.time()
            nx.minimum_cut_value(G1, G1S, G1T)
            g1_end_time = time.time() - g1_start_time
            totalTests.append(Test(0, 2, logarithmOrder, nx.density
    (G1), g1_end_time))

            # GRAFO 2
            g1_start_time = time.time()
            nx.maximum_flow(G2, G2S, G2T)
            g1_end_time = time.time() - g1_start_time
            totalTests.append(Test(1, 0, logarithmOrder, nx.density
    (G2), g1_end_time))

            g1_start_time = time.time()
            nx.maximum_flow_value(G2, G2S, G2T)
            g1_end_time = time.time() - g1_start_time
            totalTests.append(Test(1, 1, logarithmOrder, nx.density
    (G2), g1_end_time))

            g1_start_time = time.time()
```

```
            nx.minimum_cut_value(G2, G2S, G2T)
            g1_end_time = time.time() - g1_start_time
            totalTests.append(Test(1, 2, logarithmOrder, nx.density(G2), g1_end_time))

            # GRAFO 3
            g1_start_time = time.time()
            nx.maximum_flow(G3, G3S, G3T)
            g1_end_time = time.time() - g1_start_time
            totalTests.append(Test(2, 0, logarithmOrder, nx.density(G3), g1_end_time))

            g1_start_time = time.time()
            nx.maximum_flow_value(G3, G3S, G3T)
            g1_end_time = time.time() - g1_start_time
            totalTests.append(Test(2, 1, logarithmOrder, nx.density(G3), g1_end_time))

            g1_start_time = time.time()
            nx.minimum_cut_value(G3, G3S, G3T)
            g1_end_time = time.time() - g1_start_time
            totalTests.append(Test(2, 2, logarithmOrder, nx.density(G3), g1_end_time))
print(totalTests.__sizeof__())

data = pd.DataFrame.from_records([s.to_dict() for s in totalTests])

#ANOVA
formula1 = 'time ~ C(generation)'
formula2 = 'time ~ C(algorithm)'
formula3 = 'time ~ C(nodes)'
formula4 = 'time ~ C(density)'
formula5 = 'time ~ C(generation)*C(algorithm)'
formula6 = 'time ~ C(generation)*C(nodes)'
formula7 = 'time ~ C(generation)*C(density)'
formula8 = 'time ~ C(algorithm)*C(nodes)'
formula9 = 'time ~ C(algorithm)*C(density)'
formula10 = 'time ~ C(nodes)*C(density)'
formula11 = 'time ~ C(generation)*C(algorithm)*C(nodes)'
formula12 = 'time ~ C(generation)*C(algorithm)*C(density)'
formula13 = 'time ~ C(algorithm)*C(nodes)*C(density)'
formula14 = 'time ~ C(generation)*C(algorithm)*C(nodes)*C(density)'
model1 = ols(formula1, data).fit()
model2 = ols(formula2, data).fit()
model3 = ols(formula3, data).fit()
model4 = ols(formula4, data).fit()
model5 = ols(formula5, data).fit()
model6 = ols(formula6, data).fit()
model7 = ols(formula7, data).fit()
model8 = ols(formula8, data).fit()
model9 = ols(formula9, data).fit()
model10 = ols(formula10, data).fit()
model11 = ols(formula11, data).fit()
model12 = ols(formula12, data).fit()
model13 = ols(formula13, data).fit()
model14 = ols(formula14, data).fit()
anova_table1 = anova_lm(model1, typ=2)
```

```
anova_table2 = anova_lm(model2, typ=2)
anova_table3 = anova_lm(model3, typ=2)
anova_table4 = anova_lm(model4, typ=2)
anova_table5 = anova_lm(model5, typ=2)
anova_table6 = anova_lm(model6, typ=2)
anova_table7 = anova_lm(model7, typ=2)
anova_table8 = anova_lm(model8, typ=2)
anova_table9 = anova_lm(model9, typ=2)
anova_table10 = anova_lm(model10, typ=2)
anova_table11 = anova_lm(model11, typ=2)
anova_table12 = anova_lm(model12, typ=2)
anova_table13 = anova_lm(model13, typ=2)
anova_table14 = anova_lm(model14, typ=2)
#anova_table1.to_csv('anova1.xls',sep='\t')
#anova_table2.to_csv('anova2.xls',sep='\t')
#anova_table3.to_csv('anova3.xls',sep='\t')
#anova_table4.to_csv('anova4.xls',sep='\t')
#anova_table5.to_csv('anova5.xls',sep='\t')
#anova_table6.to_csv('anova6.xls',sep='\t')
#anova_table7.to_csv('anova7.xls',sep='\t')
#anova_table8.to_csv('anova8.xls',sep='\t')
#anova_table9.to_csv('anova9.xls',sep='\t')
#anova_table10.to_csv('anova10.xls',sep='\t')
#anova_table11.to_csv('anova11.xls',sep='\t')
#anova_table12.to_csv('anova12.xls',sep='\t')
#anova_table13.to_csv('anova13.xls',sep='\t')
#anova_table14.to_csv('anova14.xls',sep='\t')
#data.to_csv('data.xls', sep='\t')

#MATRIZ CORRELACION
data = data.iloc[:, :-1]
data.columns = ['algoritmo','densidad','generacion',
                'orden']
corr = data.corr()
print(corr.columns)
fig, ax = plt.subplots(figsize=(6, 6))
ax.matshow(corr)
plt.xticks(range(len(corr.columns)), corr.columns)
plt.yticks(range(len(corr.columns)), corr.columns)
cax = ax.matshow(corr,cmap='OrRd', vmin=-1, vmax=1, aspect='equal',
    origin='lower')
fig.colorbar(cax)
plt.show()
```

Este proceso se realizó en una laptop con las siguientes características:

- Procesador: Intel Core i7-7500U 2.7GHz
- Memoria RAM: 16GB
- Sistema Operativo: Windows 10 64 bits

2. Resultados

Se obtuvieron los siguientes resultados de las replicas, una matriz de correlación en la figura 1 y las pruebas estadísticas de ANOVA para determinar los

efectos de los factores:

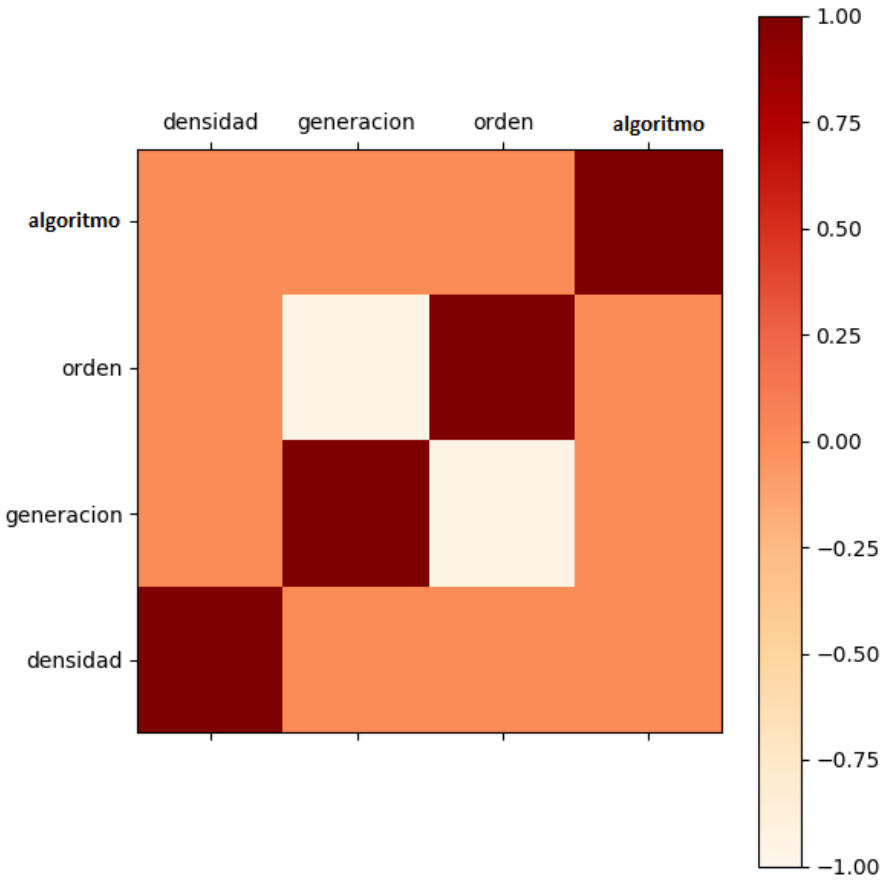

Figura 1: Matriz de correlación.

ANOVA	Grados de Libertad	Valor P
C(generation)	2	6.06E-102
C(algoritmo)	2	0.155326227
C(orden)	3	1.41E-221
C(densidad)	11	0
C(generation):C(algoritmo)	4	0.49355476
C(generation):C(orden)	6	0
C(algoritmo):C(orden)	6	0.338715638
C(generation):C(densidad)	22	0.99244162
C(algoritmo):C(densidad)	22	2.37E-77
C(orden):C(densidad)	33	0.999058669
C(generation):C(algoritmo):C(orden)	12	7.24E-27
C(generation):C(algoritmo):C(densidad)	44	0.997362183
C(generation):C(orden):C(densidad)	66	7.24E-27
C(algoritmo):C(orden):C(densidad)	66	0.973080866
C(generation):C(algoritmo):C(orden):C(densidad)	132	0.999714748
Residual	1764	

3. Conclusiones

Como conclusión de esta investigación, se tiene que cada uno de los factores de algoritmo, orden, generación de grafo y densidad afectan directamente con el tiempo de ejecución de los algoritmos, además que, en combinación, todos excepto la combinación de orden-generación afectan al rendimiento del algoritmo. La tabla ANOVA respalda los resultados con los valores de P dados, y además muestran que existe una interacción entre los factores de generación-orden-densidad y generación-algoritmo-orden.

Referencias

[1] Python Software Foundation Versión 3.7.2. https://www.python.org/.

[2] NetworkX developers con última actualización el 19 de Septiembre 2018. https://networkx.github.io/documentation/stable/index.html.

[3] NetworkX developers Versión 2.0. https://networkx.github.io/.

[4] The Matplotlib development team Versión 3.0.2. https://matplotlib.org/.

[5] Patlán Castillo Jesús Angel. Repositorio Optimización Flujo en Redes. https://github.com/JAPatlanC/Flujo-Redes.