Tarea 4 Optimización de Flujo en Redes

Orlando Lázaro Ruiloba Torres 5270

2 de abril de 2019

1. Breve explicación del objetivo que se persigue

Para la realización de este trabajo se seleccionaron tres métodos generadores de grafos, de los cuales en las secciones siguientes se expone una breve explicación, a partir de estos se generaron 10 grafos de cuatro órdenes distintos en escala logarítmica, a los cuales se les asignaron pesos no negativos y normalmente distribuidos con el fin de poder implementar tres de los algoritmos de networkx para flujo máximo seleccionados, los cuales se ejecutaron cinco veces para los cinco pares fuente - sumidero elegidos, con el fín de determinar mediante el uso de técnicas estadísticas el efecto o interacción existente por parte los métodos generadores, los algoritmos usados, el orden y la densidad de los grafos en el tiempo de ejecución.

A continuación se ofrece una breve explicación de cada uno de los métodos y algoritmos seleccionados:

2. Método generador de grafos: Árbol lleno r-ario

Este método crea un árbol r-ario completo de n vértices. Todos los vértices que no son hojas tienen exactamente r hijos y todos los niveles están llenos excepto por alguna posición más a la derecha del nivel inferior (si falta una hoja en el nivel inferior, entonces también están todas las hojas a su derecha [2].

3. Método generador de grafos: Grafo completo

Este método devuelve un grafo completo K, con n vértices, en el caso de que n sea un número entero, los vértices son de rango n, por otro lado si n es un contenedor de vértices, estos aparecen en el grafo [3]. Un grafo completo, es un grafo en el que cada vértice comparte una arista con cada uno de los otros vértices. Si se trata de un grafo dirigido, las aristas deben existir siempre en ambas direcciones [7].

4. Método generador de grafos: Grafo rueda

Este método devuelve un grafo rueda. El grafo de la rueda consiste en un vértice central conectado a un ciclo de $n - 1$ vértices [4] y para el cual cada vértice del grafo en el ciclo está conectado a otro [8].

5. Algoritmo: Valor de flujo máximo

Este algoritmo consiste en determinar cuál es la tasa mayor a la cual un determinado material puede ser transportado de la fuente al sumidero sin violar ninguna restricción de capacidad [1].

6. Algoritmo: Corte mínimo

Este algoritmo calcula el valor del corte mínimo y de la partición de vértice. Utiliza el teorema de corte mínimo de flujo máximo, es decir, la capacidad de un corte de capacidad mínima es igual al valor de flujo de un flujo máximo [5].

7. Algoritmo: Valor de corte mínimo

Este algoritmo es muy similar al anterior, pero tiene la tipicidad de que solo calcula el valor del corte mínimo. Utiliza el teorema de corte mínimo de flujo máximo, es decir, la capacidad de un corte de capacidad mínima es igual al valor de flujo de un flujo máximo [6].

8. Discusión de conclusiones

A continuación se muestran las cuatro gráficas de caja necesarias para efectuar el análisis del comportamiento del efecto que tienen sobre el tiempo de ejecución, los tres métodos generadores de grafos, los tres algoritmos seleccionados, el orden de cada grafo y la densidad de los mismos:

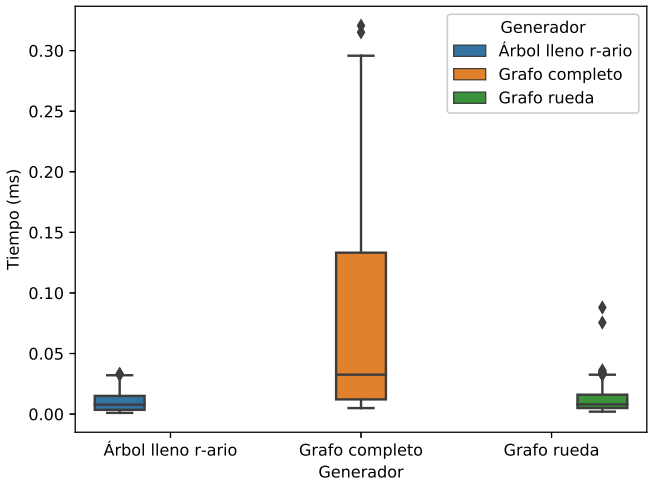

Figura 1: Efecto de los métodos generadores de grafos seleccionados sobre el tiempo de ejecución

Como se puede apreciar en la Figura 1, el método generador de grafos que más tardó en ejecutarse fue el de Grafo completo, siendo este el que mayor efecto tuvo sobre el tiempo de ejecución, en los otros dos casos los tiempos fueron muy similares.

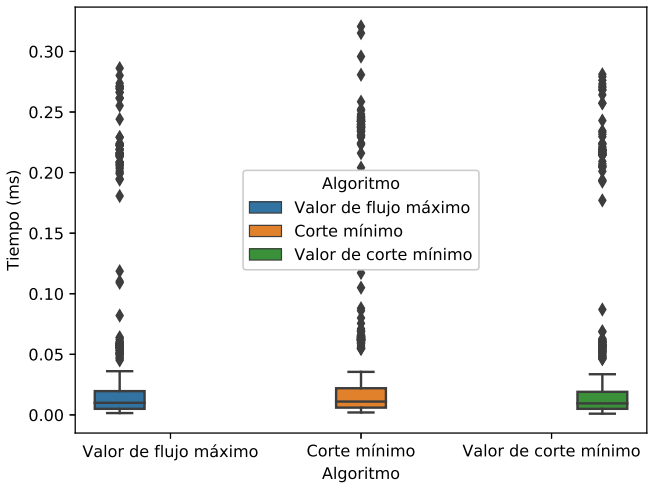

Figura 2: Efecto de los algoritmos seleccionados sobre el tiempo de ejecución

A partir de la Figura 2, se puede concluir que todos los algoritmos seleccionados mostraron tiempos de ejecución similares, por lo que a través de la realización del análisis estadístico (ANOVA) que se realiza con posterioridad se podra concluir de manera más certera el posible efecto.

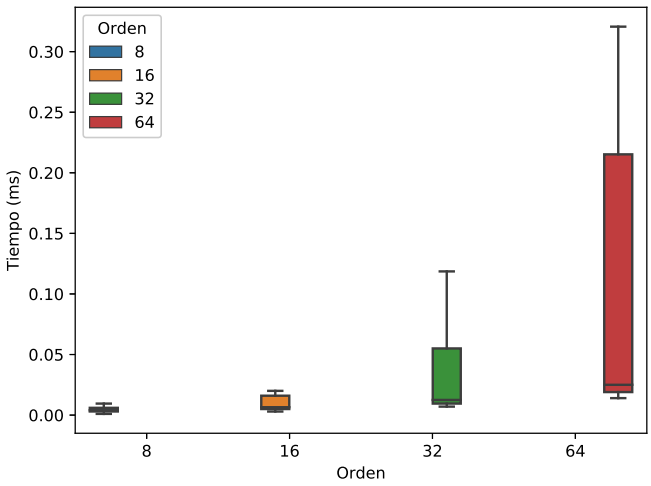

Figura 3: Efecto del orden del grafo sobre el tiempo de ejecución

Como se puede observar en la Figura 3, a medida que aumenta el orden del grafo aumenta también el tiempo de ejecución, por lo que se puede concluir que son directamente proporcionales y que definitivamente el orden seleccionado va a tener un efecto considerable en el tiempo de ejecución.

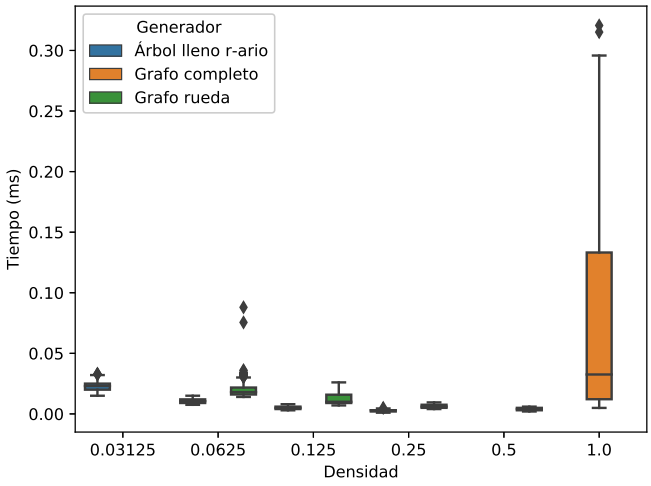

Figura 4: Efecto de la densidad del grafo sobre el tiempo de ejecución

Como se puede apreciar a partir de la figura anterior, la variación de los tiempos de ejecución es muy pequeña, sin embargo cuando la densidad llega a 1, el valor de este se incrementa de forma considerable, lo que permite afirmar que la densidad del grafo si va a repercutir en el tiempo de ejecución.

Seguidamente se muestra un cuadro en el que se exponen los resultados del análisis de varianza (ANOVA):

	sum sq	**df**	**F**	**PR(>F)**
Generador	1.394225e-10	2.0	3.931140e-07	1.000000
Algoritmo	4.165406e-03	2.0	1.174473e+01	0.000009
Orden	-1.289626e-11	1.0	-7.272428e-08	1.000000
Densidad	1.511910e-10	1.0	8.525926e-07	0.999263
Generador:Algoritmo	2.385267e-04	4.0	3.362736e-01	0.853637
Generador:Orden	9.396538e-07	2.0	2.649437e-03	0.997354
Generador:Densidad	1.879308e-06	2.0	5.298874e-03	0.941979
Algoritmo:Densidad	6.478017e-04	2.0	1.826534e+00	0.161272
Orden:Densidad	9.396538e-07	1.0	5.298874e-03	0.941979
Residual	3.160036e-01	1782.0		

Cuadro 1: Resultados obtenidos ANOVA

A partir de los resultados expuestos anteriormente se puede concluir que para el caso de el método generador, el orden del grafo y la densidad, como su p valor > 0.05, el tiempo de ejecución va a sufrir variaciones en dependencia del método elegido, el orden asignado y la densidad que posea el grafo en cuestión, lo cual no sucede para el caso del algoritmo seleccionado, el cual no va a tener ninguna injerencia o efecto sobre el tiempo de ejecución.

Referencias

[1] Agustín J. González. Estructura de datos y algoritmos. Last access in 04-01-2019 to http://profesores.elo.utfsm.cl/ agv/elo320/01and02/redesDeFlujo/maximumFlow.pdf, 2002.

[2] Networkx. Networkx generators classic full_rary_tree. Last access in 04-01-2019 to http://networkx.github.io/documentation/stable/reference /generated/networkx.generators.classic.full_rary_tree.html#networkx.generators.classic.full_rary_tree, 2018.

[3] Networkx. Networkx generators classic complete_graph. Last access in 04-01-2019 to http://networkx.github.io/documentation/stable/reference /generated/networkx.generators.classic.complete_graph.html#networkx.generators.classic.complete_graph, 2018.

[4] Networkx. Networkx generators classic wheel_graph. Last access in 04-01-2019 to http://networkx.github.io/documentation/stable/reference /generated/networkx.generators.classic.wheel_graph.html#networkx.generators.classic.wheel_graph, 2018.

[5] Networkx. Networkx algorithms flow minimum_cut. Last access in 04-01-2019 to http://networkx.github.io/documentation/stable/reference /algorithms/generated/networkx.algorithms.flow.minimum_cut.html#networkx.algorithms.flow.minimum_cut, 2018.

[6] Networkx. Networkx algorithms flow minimum_cut_value. Last access in 04-01-2019 to http://networkx.github.io/documentation/stable/reference /algorithms/generated/networkx.algorithms.flow.minimum_cut_value.html#networkx.algorithms.flow.minimum_cut_value, 2018.

[7] Tim Shearouse. Class completegraphgenerator. Last access in 04-01-2019 to http://jgrapht.org/javadoc/org/jgrapht/generate/CompleteGraphGenerator.html, 2018.

[8] MathWorld Team. Wheel graph. Last access in 04-01-2019 to http://mathworld.wolfram.com/WheelGraph.html, 2019.