Complejidad asintótica experimental

Brenda Yaneth Sotelo Benítez
5705

3 de junio de 2019

En este trabajo se realiza la medición de tiempo de ejecución de tres métodos de generación de grafos de orden 8, 16, 32, 64 a base 2 con tres implementaciones de algoritmos de flujo máximo. Se generaron diez grafos distintos de cada orden con la asignación de pesos positivos normalmente distribuidos a las aristas con media $\mu = 10$ y desviación estándar $\sigma = 3$ para poder ser utilizados en las instancias del problema de flujo máximo.

Los algoritmos seleccionados se ejecutan con cinco diferentes pares de fuente-sumidero y se obtuvieron por medio de la librería Networkx [1] y Matplotlib [2] de `Python` [3] para la generación de grafos y guardar imágenes en el formato *eps* respectivamente. Además se hace uso de la librería Numpy [4] para la suma de tiempos, cálculo de tiempos promedios y desviaciones estándar y Pandas [5] para el tratamiento de datos. El código empleado se obtuvo consultando documentación oficial [6].

Se presenta una breve descripción de los algoritmos seleccionados, visualización de los grafos, fragmentos relevantes de código y una sección de resultados donde se muestra un análisis de varianza para saber si los efectos entre el algoritmo generador, algoritmo de flujo máximo, orden y la densidad del grafo con el tiempo de ejecución son estadísticamente significativos.

Las pruebas se han realizado en una PC con procesador Intel Core i3 2.00 GHz y 8.00 GB de RAM.

Métodos generadores de grafos

La elección de los siguientes métodos se debe a que la visualización de cada uno es interesante, sobre todo los que generan grafos n-dimensionales, por la aplicación práctica que tienen, simplicidad y variedad.

1. `Complete_graph`. Tiene como parámetros el número de nodos que se desea y el tipo de grafo, que toma por defecto un grafo simple no dirigido.

   La figura 1 muestra una visualización de este generador.

```
G = nx.complete_graph(30)
nx.draw_networkx(G, node_color='yellow', edge_color='g',
    with_labels=False, alpha=0.8)
```

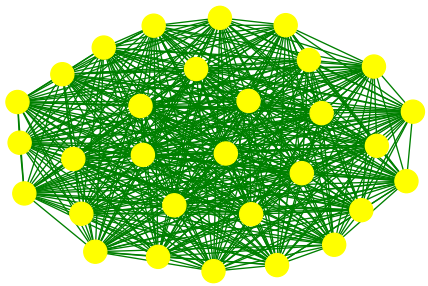


Figura 1: *Complete graph*

2. `Wheel_graph`. El grafo de la rueda tiene un nodo central que está conectado a un ciclo de $(n-1)$ nodos. Recibe como parámetros el número de nodos, el tipo de grafo que se desea crear, el cual tiene como opción predeterminada un grafo simple no dirigido. Las etiquetas de los nodos van de 0 a $n-1$.

   Se presenta la visualización del generador en la figura 2.

```
G = nx.wheel_graph(10)
nx.draw_networkx(G, node_color='b', edge_color='c', with_labels
    =False, node_size=20)
```

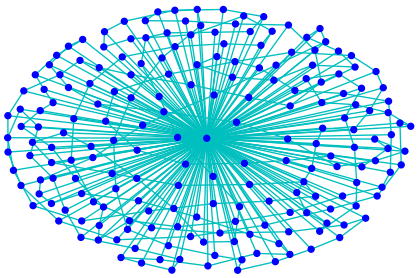


Figura 2: *Wheel graph*

3. `Grid_graph`. Recibe como principal parámetro la cantidad de nodos que puede ser una lista, tupla o un entero y devuelve un grafo de cuadrícula n-dimensional.

   La figura 3 muestra una visualización de este generador.

```
G = nx.grid_graph(dim=[8,8])
nx.draw_networkx(G, node_color='r', edge_color='k', with_labels
    =False, node_size=80)
```

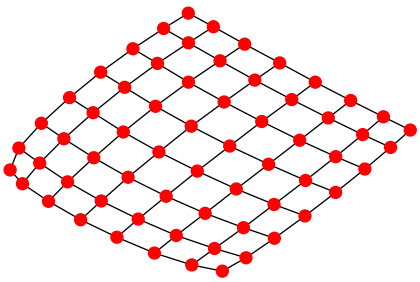


Figura 3: *Grid graph*

4. `Circular_ladder_graph`. Devuelve un grafo de escalera circular, que consiste en dos ciclos de n nodos en donde cada uno de los n pares de nodos se unen por una arista. Toma como dato de entrada la cantidad de nodos y las etiquetas de estos son los números enteros de 0 a $n-1$.

   La figura 4 muestra una visualización de este generador.

```
G = nx.circular_ladder_graph(20)
nx.draw_networkx(G, node_color='b', edge_color='k', with_labels
    =False, node_size=80)
```

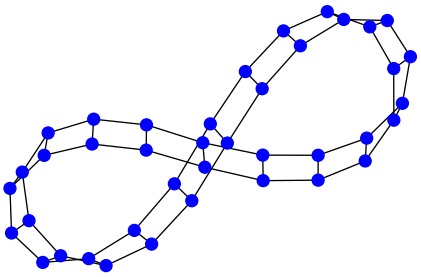


Figura 4: *Circular ladder graph*

Algoritmos de flujo máximo

La elección de los siguientes algoritmos se debe a la variedad que hay para calcular el valor del flujo de un grafo, que reciben parámetros similares y retornan diferentes datos.

1. `Maximum_flow_value`. Encuentra y devuelve el valor del flujo máximo donde cada arista debe tener una capacidad, que en caso de no tenerla toma por defecto una capacidad infinita, recibe un nodo fuente y un nodo sumidero y una función para calcular el flujo máximo. El algoritmo no es compatible con multigrafos.

```
G = nx.DiGraph()
G.add_edge(1, 2, capacity=5)
G.add_edge(2, 3, capacity=10)
G.add_edge(3, 4, capacity=20)
G.add_edge(1, 5, capacity=30)
G.add_edge(5, 6, capacity=15)
G.add_edge(6, 4, capacity=15)
G.add_edge(2, 5, capacity=20)
G.add_edge(5, 3, capacity=10)
G.add_edge(3, 6, capacity=5)

flow_value = nx.maximum_flow_value(G, 1, 6)
```

2. `Minimum_cut`. Recibe como parámetros principales la capacidad y si este dato se omite se considera que tiene una capacidad infinita , nodo fuente y sumidero para el flujo y una función para calcular el flujo máximo. Devuelve el valor del corte mínimo y la partición de los nodos que definen el corte mínimo.

```
G = nx.DiGraph()
G.add_edge(1, 2, capacity=5)
G.add_edge(2, 3, capacity=10)
G.add_edge(3, 4, capacity=20)
G.add_edge(1, 5, capacity=30)
G.add_edge(5, 6, capacity=15)
G.add_edge(6, 4, capacity=15)
G.add_edge(2, 5, capacity=20)
G.add_edge(5, 3, capacity=10)
G.add_edge(3, 6, capacity=5)

cut_value, partition = nx.minimum_cut(G, 1, 6)
```

3. `Minimum_cut_value`. Este algoritmo devuelve solamente el valor que tiene el corte mínimo. Tiene como datos de entrada el nodo fuente y nodo sumidero, una función para calcular el flujo máximo y la capacidad de las aristas que por defecto se considera que tiene una capacidad infinita.

```
G = nx.DiGraph()
G.add_edge(1, 2, capacity=5)
G.add_edge(2, 3, capacity=10)
G.add_edge(3, 4, capacity=20)
G.add_edge(1, 5, capacity=30)
G.add_edge(5, 6, capacity=15)
G.add_edge(6, 4, capacity=15)
G.add_edge(2, 5, capacity=20)
G.add_edge(5, 3, capacity=10)
G.add_edge(3, 6, capacity=5)

cut_value = nx.minimum_cut_value(G, 1, 6)
```

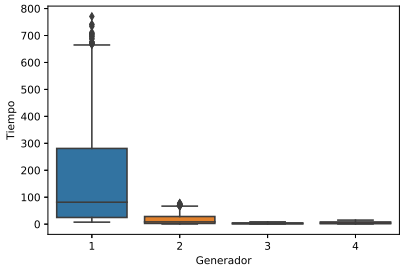
Resultados

Los resultados del análisis de varianza de los efectos simples se muestran en el cuadro 1, donde los factores que presentan una relación significativa con el tiempo de ejecución son la densidad, generador de grafo y orden, es decir, que tienen un p-valor menor a 0.05 (nivel de significancia) y para los que la hipótesis nula de que las medias son iguales se rechaza. Mientras que el factor algoritmo no es significativo con el tiempo de ejecución.

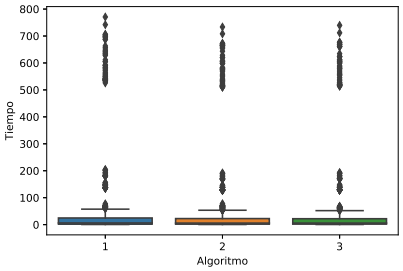
Cuadro 1: *Efectos simples*

Factor	P**-valor**
Algoritmo	0.316
Densidad	0.000
Generador	0.000
Orden	0.000

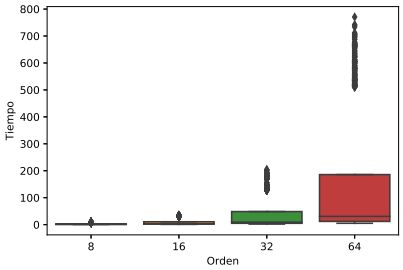
Por otro lado, en la figura 5 se muestran los diagramas de caja y bigotes para cada nivel de los factores donde se observa que el generador 1 es el que mayor tiempo consume a comparación del generador 2, los tres tipos de algoritmos parecen tener tiempos de ejecución similares, conforme va creciendo el orden de los grafos el tiempo de ejecución crece también y que a menor densidad el tiempo aumenta.



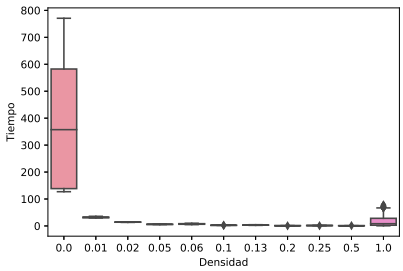
(a) Generador de grafo contra tiempo



(b) Algoritmo contra tiempo



(c) Orden del grafo contra tiempo



(d) Densidad del grafo contra tiempo

Figura 5: *Diagramas de caja y bigotes de los factores.*

Además, en el cuadro 2 se muestran los efectos de interacción para cada pareja de factor de los factores mencionados en el cuadro 1, donde se puede concluir que las interacciones generador-densidad, generador-orden y orden-densidad tienen relación significativa, es decir, que tienen un p-valor menor a 0.05 (nivel de significancia), por lo que la combinación que se tome de cada uno de los factores importa para el valor del tiempo de ejecución. Mientras que las interacciones

algoritmo-densidad, algoritmo-orden y generador-orden no tienen relación significativa con el tiempo dado que el p-valor es mayor a 0.05, por lo que no importa que combinación se tenga de los factores.

Cuadro 2: *Efectos de interacción*

Factor	**P-valor**
Algoritmo-Densidad	0.785
Algoritmo-Orden	0.417
Generador-Algoritmo	0.311
Generador-Densidad	0.000
Generador-Orden	0.000
Orden-Densidad	0.000

Referencias

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

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

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

[4] NumPy developers Versión 1.16.2. `https://networkx.github.io`.

[5] Augspurger T. and the pandas core team Versión 0.24.2. `https://pandas.pydata.org/`.

[6] NetworkX developers con última actualización el 19 de Septiembre 2018. `https://networkx.github.io/documentation/networkx-1.10/reference/drawing.html`.