

# Optimización de flujo en redes

5175

Tarea #3

19 de marzo de 2019

Se utilizan algoritmos para grafos de NetworkX, implementando un código Python para ejecutar los siguientes algoritmos:

- `all_shortest_paths`
- `betweenness_centrality`
- `dfs_tree`
- `greedy_color`
- `max_weight_matching`

luego se aplica una cantidad moderada de iteraciones para crear tiempos de ejecución considerables para después hacer un análisis estadístico.

## 1. Caminos cortos

Se utiliza el primer grafo de la tarea 2 con el algoritmo `all_shortest_paths` de NetworkX, el cual encuentra todos los caminos cortos de un nodo inicial a un nodo destino, se aplica un total de iteraciones de 8 millones para alcanzar un tiempo de ejecución mayor a 5 segundos como se muestra a continuación:

```
1 tiempoAngell = []
2 for i in range(30):
3     start = tm.time()
4     for x in range(8000000):
5         nx.all_shortest_paths(G, source='C', target='B')
6     end = tm.time()
7     tiempoAngell.append(end - start)
```

La figura 1 muestra los resultados obtenidos de la implementación, se muestra la prueba estadística de *shapiro wilk* para probar si la distribución de los datos se comportan normales, dado que el *p* valor es mayor a 0.05 se puede concluir que los datos se comportan normales, además se muestra la media y desviación estándar.

```
1 tiempoAngell = []
2 for i in range(30):
3     start = tm.time()
4     for x in range(8000000):
5         nx.all_shortest_paths(G, source='C', target='B')
6     end = tm.time()
7     tiempoAngell.append(end - start)
```

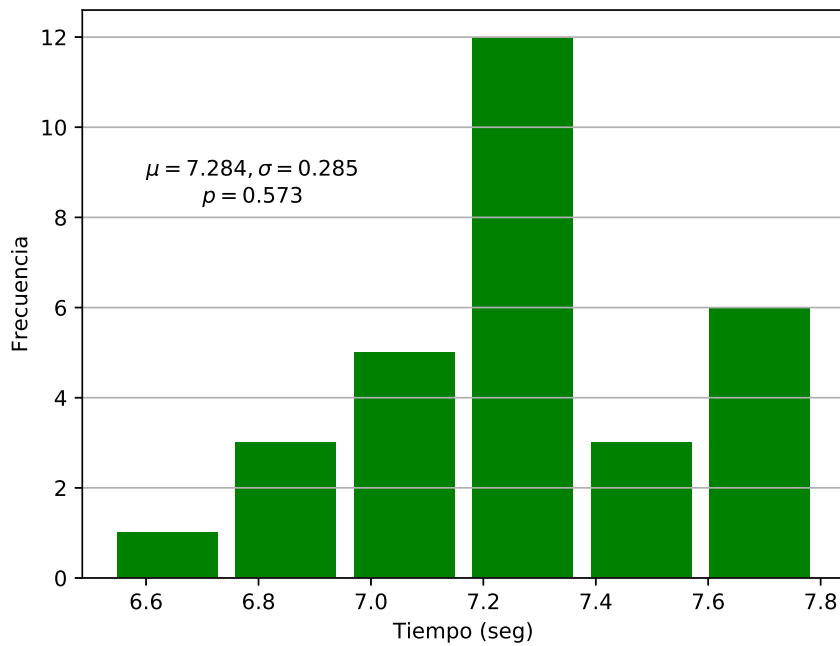


Figura 1: Histograma del tiempo de ejecución del algoritmo.

## 2. Nodos centrales

Se implementa subrutina similar a la seccion pasada con la diferencia de utilizar el algoritmo `betweenness centrality`, el cual es perfecto para encontrar caminos cortos desde un nodo inicial central hasta uno final. Se utiliza el segundo grafo de la tarea 2.

La figura 2 muestra el histograma del tiempo de ejecucion, se observa que los datos no se comportan normales debido al  $p$  valor muy pequeño de la prueba estadística.

```

1 tiempoAngel2 = []
2 for i in range(30):
3     start = tm.time()
4     for x in range(20000):
5         nx.betweenness centrality(G, normalized=True)
6     end = tm.time()
7     tiempoAngel2.append(end - start)

```

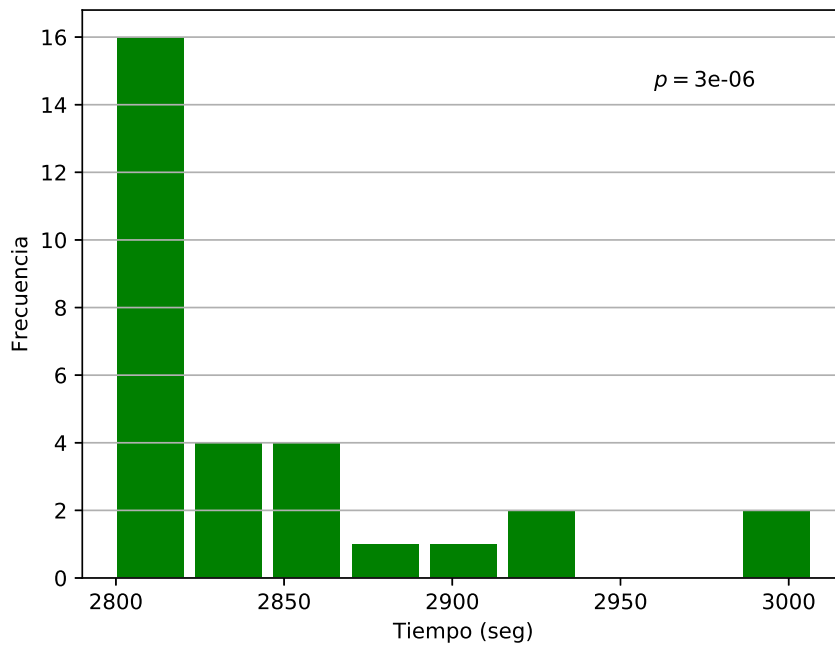


Figura 2: Histograma del tiempo de ejecución del algoritmo `betweenness centrality`.

### 3. Búsqueda en profundidad

El algoritmo `dfs_tree` construye un árbol orientado partiendo desde un nodo inicial utilizando la búsqueda a profundidad. La figura 3 muestra el histograma, se vuelve a rechazar normalidad de los datos.

```

1 tiempoAngel3 = []
2 for i in range(30):
3     start = tm.time()
4     for x in range(100000):
5         nx.dfs_tree(G, "D")
6     end = tm.time()
7     tiempoAngel3.append(end - start)

```

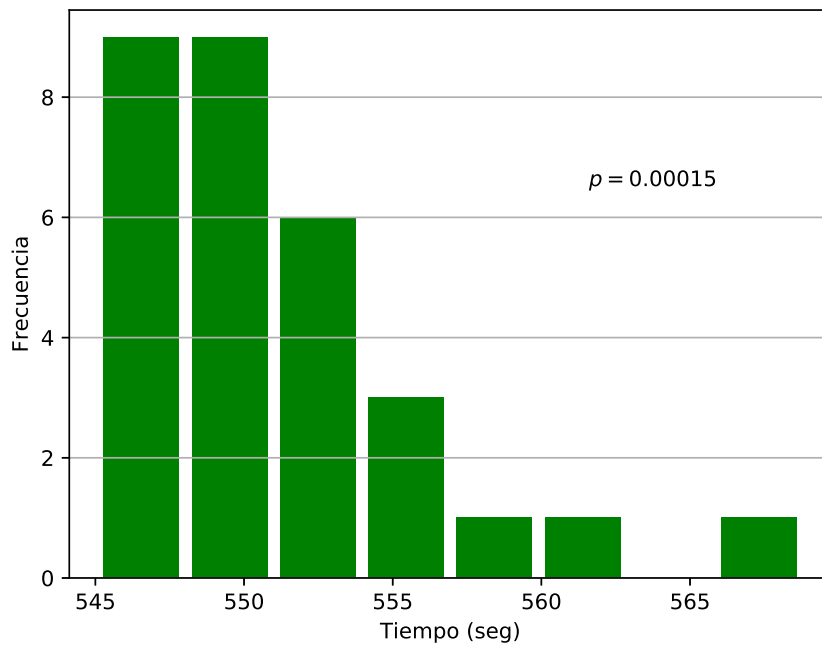


Figura 3: Histograma del tiempo de ejecución utilizando el algoritmo `dfs_tree`.

## 4. Colores

La figura 4 muestra el histograma del tiempo de ejecución de usar el algoritmo `greedy_color` el cual encuentra los colores adecuados para un grafo utilizando estrategias. Se rechaza normalidad de los datos.

```

1 tiempoAngel4 = []
2 for i in range(30):
3     start = tm.time()
4     for x in range(100000):
5         nx.greedy_color(G, strategy='largest_first')
6     end = tm.time()
7     tiempoAngel4.append(end - start)

```

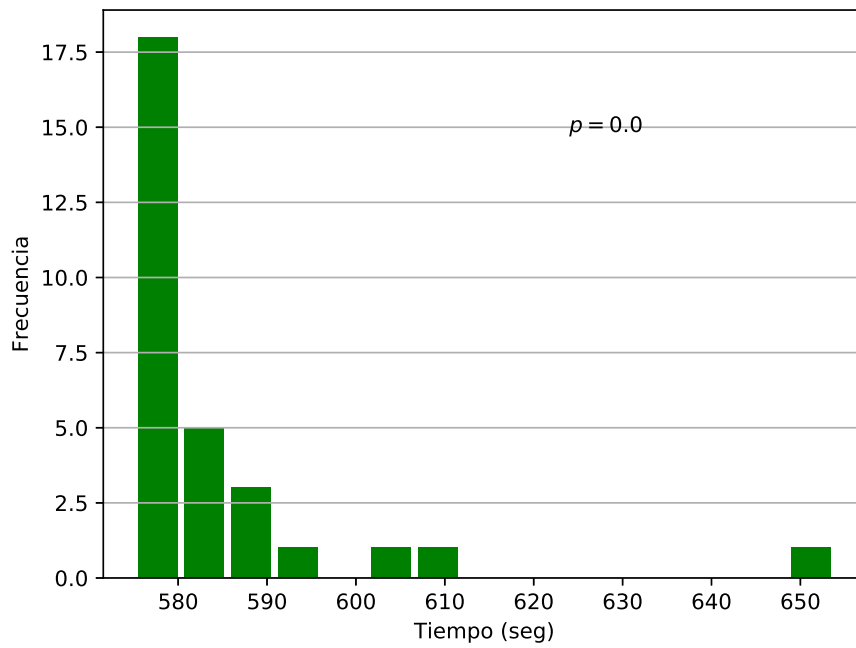


Figura 4: Tiempo de ejecución del algoritmo `greedy_color`.

## 5. Maximo peso

Utilizando el algoritmo `max_weight_matching` se consigue el histograma de la figura 5, el algoritmo consiste en encontrar el camino con el maximo peso partiendo desde un nodo inicial. Se rechaza normalidad de los datos.

```

1 tiempoAngel5 = []
2 for i in range(30):
3     start = tm.time()
4     for x in range(25000):
5         nx.max_weight_matching(G)
6     end = tm.time()
7     tiempoAngel5.append(end - start)

```

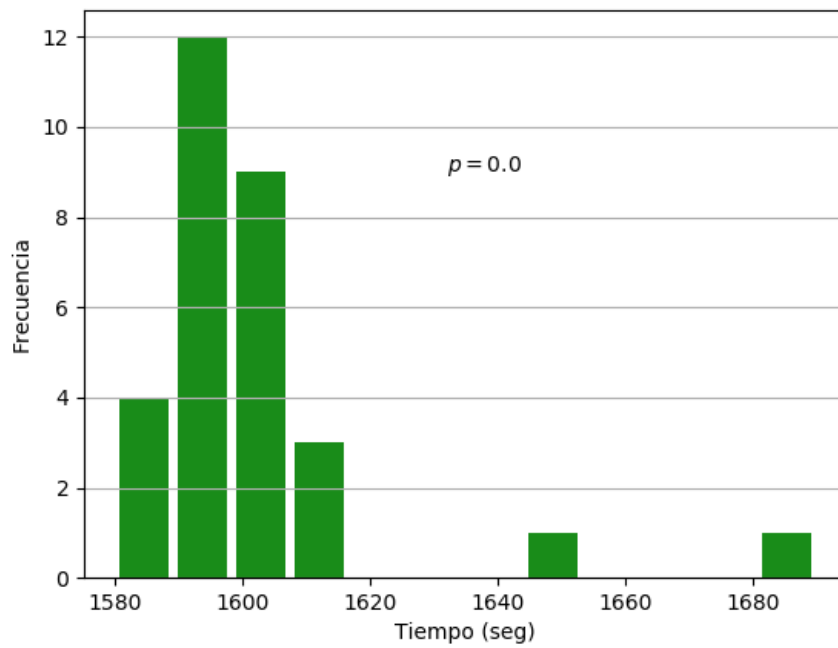


Figura 5: Histograma del tiempo de ejecución del algoritmo `max_weight_matching`.

## 6. Conclusiones

La figura 6 muestra una gráfica de dispersión de tiempo de ejecución contra cantidad de nodos de los grafos. Se observa como varían los tiempos dependiendo la cantidad de nodos en el grafo.

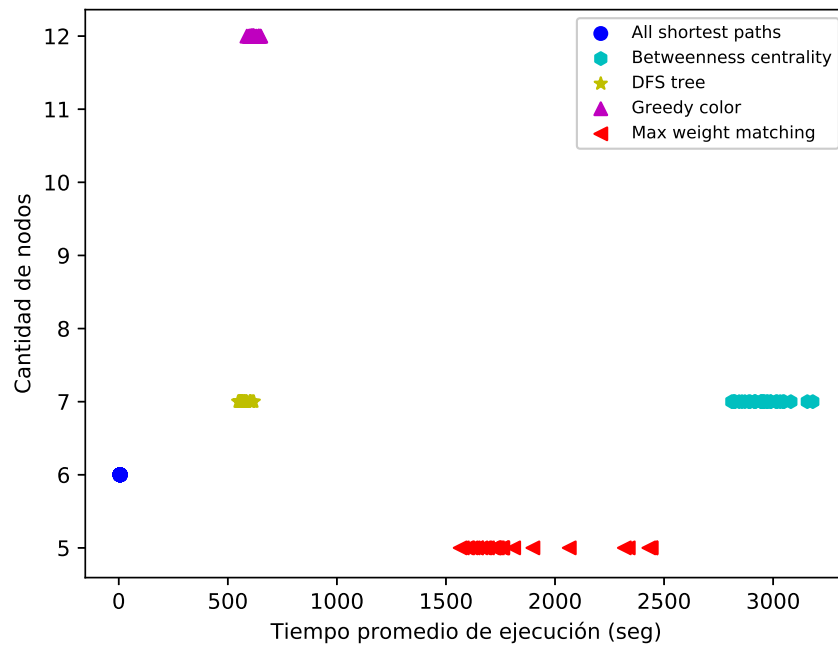


Figura 6: Gráfica de dispersión de tiempos contra cantidad de nodos del grafo.

La figura 7 muestra los resultados de los tiempos de ejecución contra la cantidad de aristas del grafo.

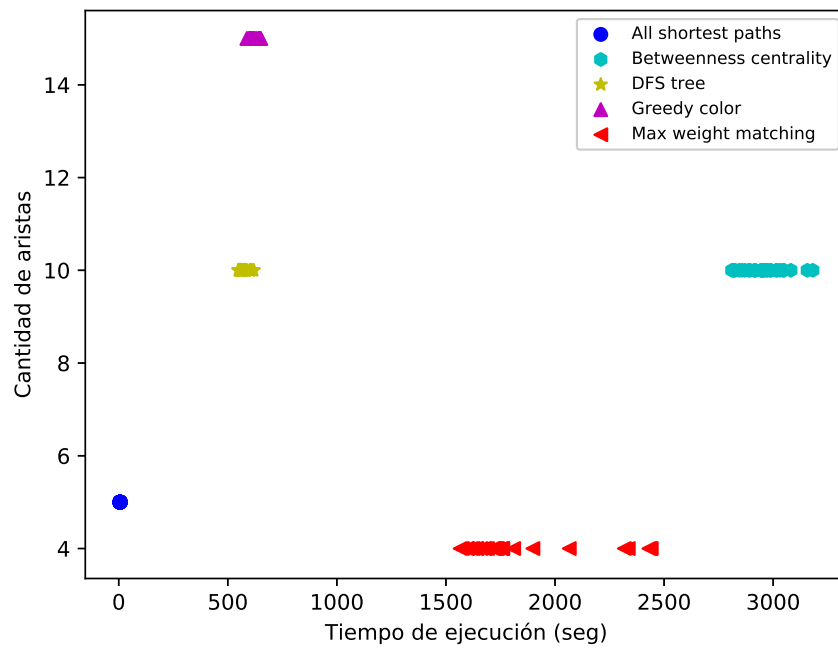


Figura 7: Gráfica de dispersión de tiempos contra cantidad de aristas del grafo.

## Referencias

- [1] SCHAEFFER E. *Optimización de flujo en redes*, 2019.  
<https://elisa.dyndns-web.com/teaching/opt/flow/>
- [2] TOMIHISA KAMADA, SATORU KAWAI. *An Algorithm for Drawing General Undirected Graphs*  
Information Processing Letters, 1988.