Optimización de flujo en redes

5171
Tarea #3

19 de marzo de 2019

Utilizando las algoritmos para grafos de NetworkX, se implementa un código en Python que ejecuta los siguientes cinco algoritmos.

1. All shortest paths
2. Betweenness centrality
3. Depth first search tree
4. Greedy color
5. Maximal weight matching

1. All shortest paths.

Este algoritmo encuentra las longitudes de las rutas más cortas entre todos los pares de vértices.

1.1. Código

```
tiempos = []
for i in range(30):
    start = tm.time()
    for x in range(8000000):
        nx.all_shortest_paths(G, source='1', target='3')
    end = tm.time()
    tiempos.append(end - start)
```

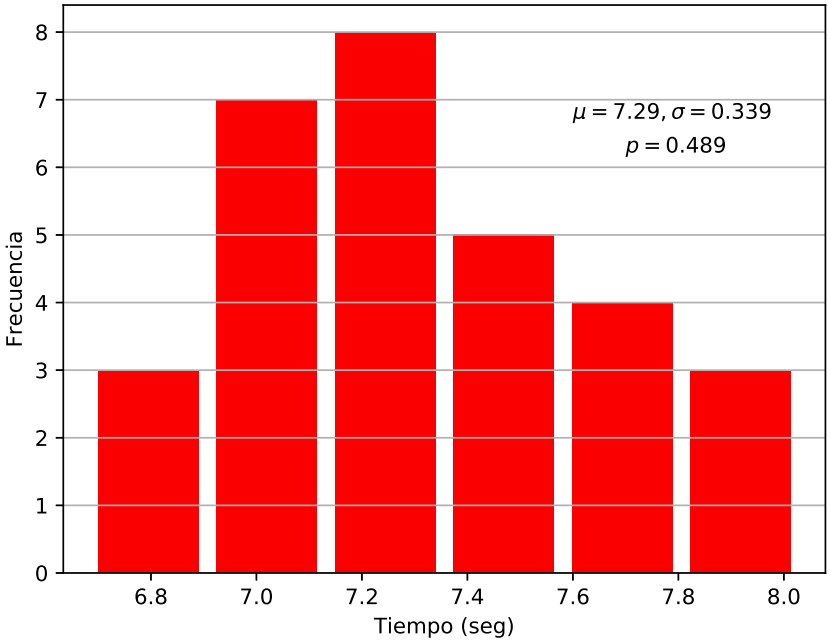

Figura 1: Histrograma.

2. Betweenness centrality

Para cada par de vértices en un gráfico conectado, existe al menos una ruta más corta entre los vértices, de manera tal que el número de bordes por los que pasa el camino se minimiza. La centralidad intermedia entre cada vértice es el número de estos caminos más cortos que pasan a través del vértice.

2.1. Código

```
tiempo2 = []
for i in range(30):
    start = tm.time()
    for x in range(8000000):
        nx.betweenness_centrality(G, normalized=True)
    end = tm.time()
    tiempo2.append(end - start)
```

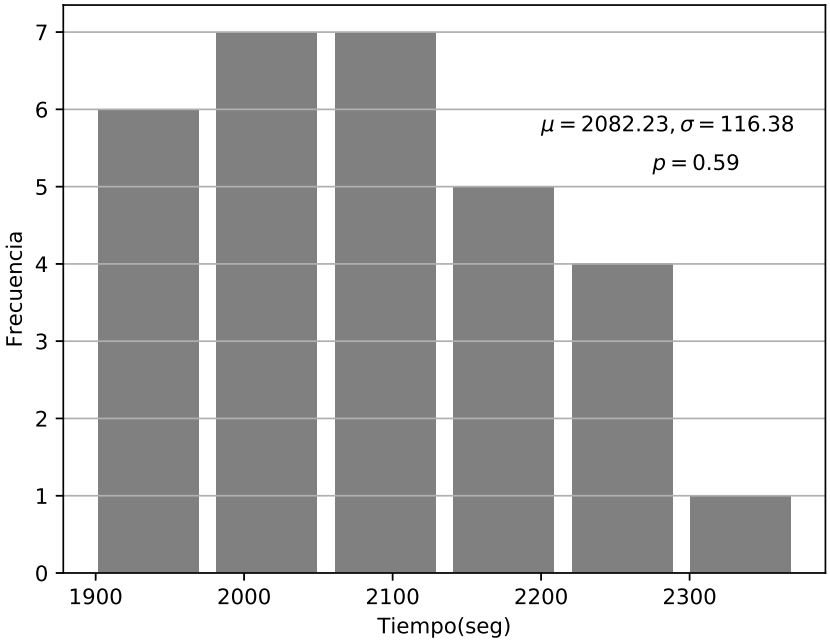

Figura 2: Histrograma.

3. Depth first search tree

Es un árbol orientado construido a partir de una fuente de búsqueda en profundidad.

3.1. Código

```
tiempo3 = []
for i in range(30):
    start = tm.time()
    for x in range(8000000):
        nx.dfs_tree(G, source=0)
    end = tm.time()
    tiempo3.append(end - start)
```

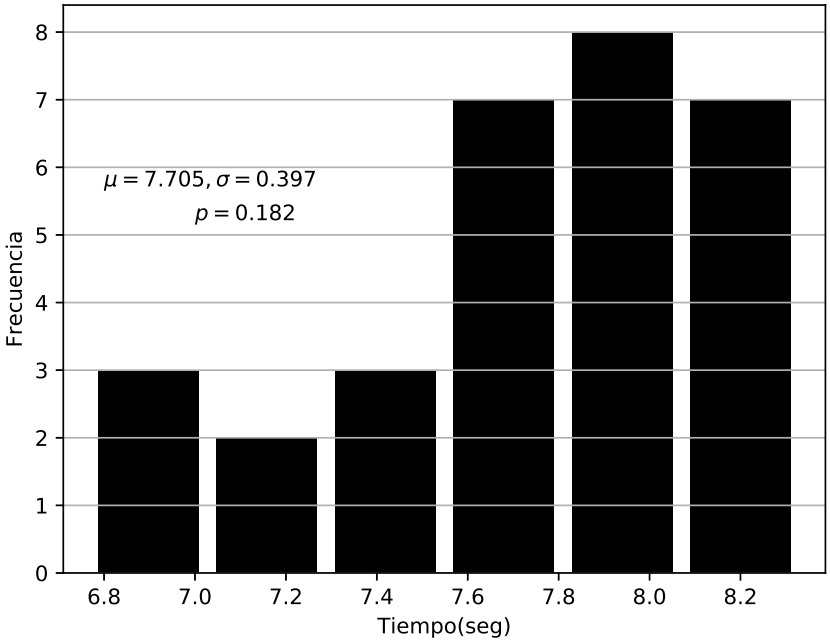

Figura 3: Histrograma.

4. Greedy color

Intenta colorear una gráfica con la menor cantidad de colores posible, donde ningún vecino de un nodo puede tener el mismo color que el nodo mismo. La estrategia dada determina el orden en que se colorean los nodos.

4.1. Código

```
tiempo4 = []
for i in range(30):
    start = tm.time()
    for x in range(8000000):
        nx.greedy_color(G, strategy='largest_first')
    end = tm.time()
    tiempo4.append(end - start)
```

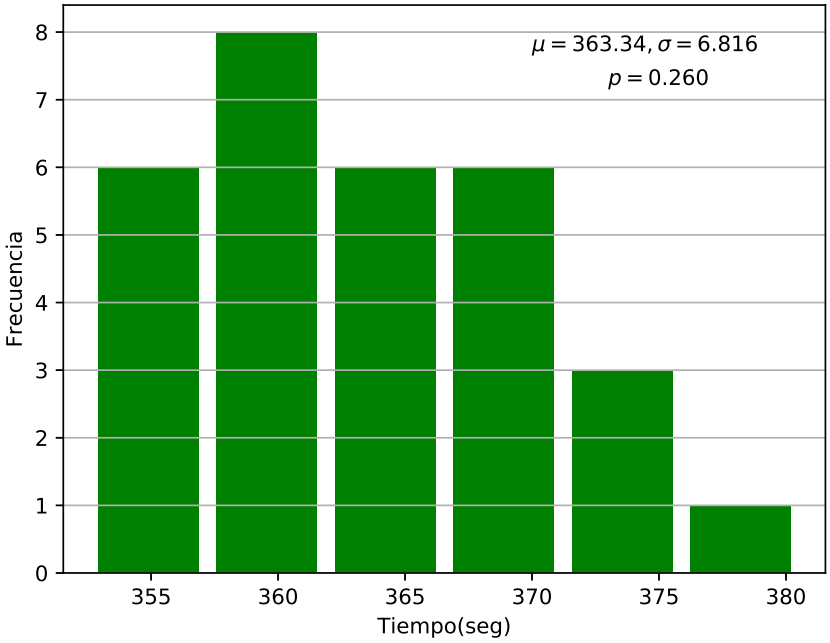

Figura 4: Histrograma.

5. Maximal weight matching

Calcula una coincidencia máxima ponderada de G. Una coincidencia es un subconjunto de bordes en los que no se produce ningún nodo más de una vez.

5.1. Código

```
tiempo5 = []
for i in range(30):
    start = tm.time()
    for x in range(8000000):
        nx.max_weight_matching(G)
    end = tm.time()
    tiempo5.append(end - start)
```

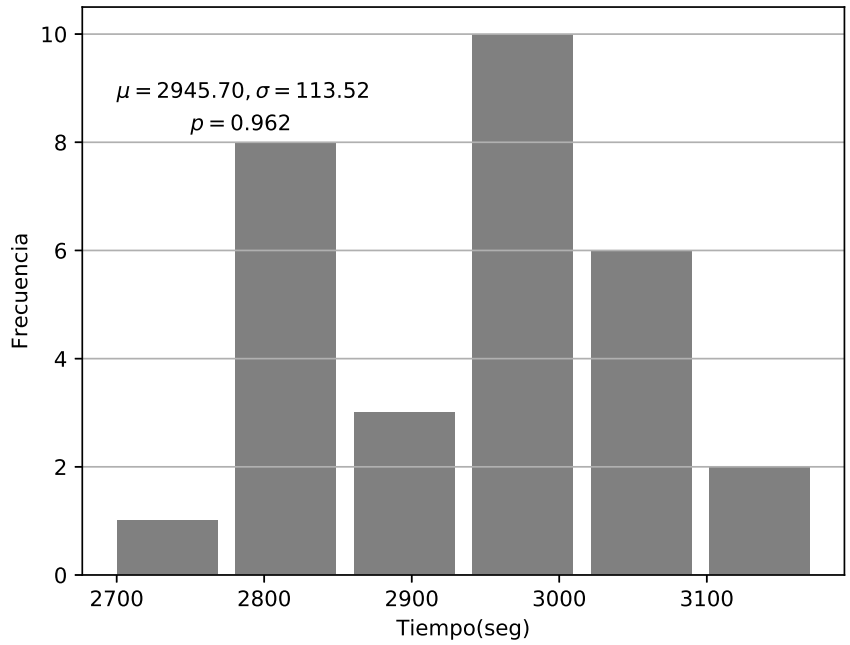

Figura 5: Histrograma.

En cada una de las figuras anteriores se les realizó a los datos la prueba de *Shapiro*, la cual arrojó normalidad para los tiempos de ejecución de la implementación de todos los algoritmos.

En las figuras 6 y 7, se observa como los algoritmos más tardados son el betweenness centrality y maximal weight matching.

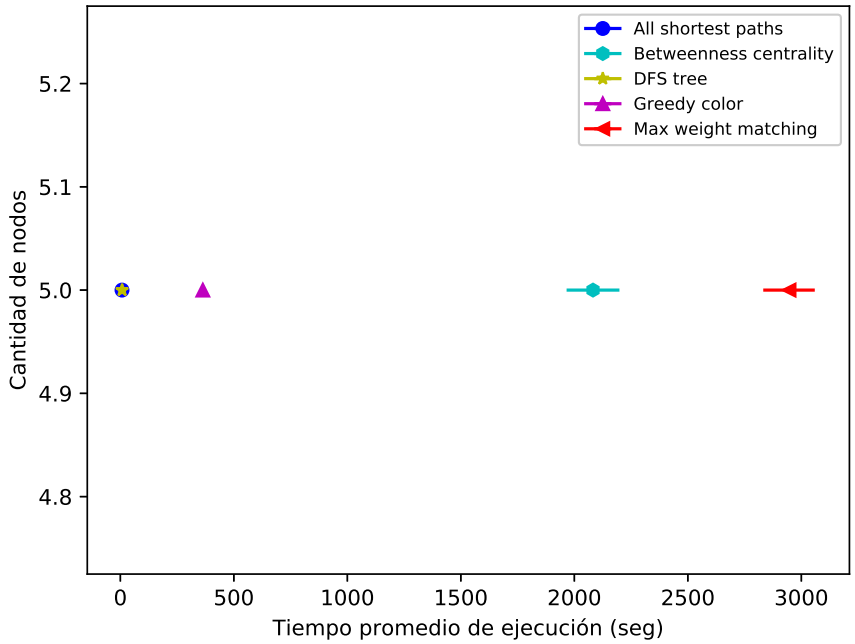

Figura 6: Diagrama de dispersión.

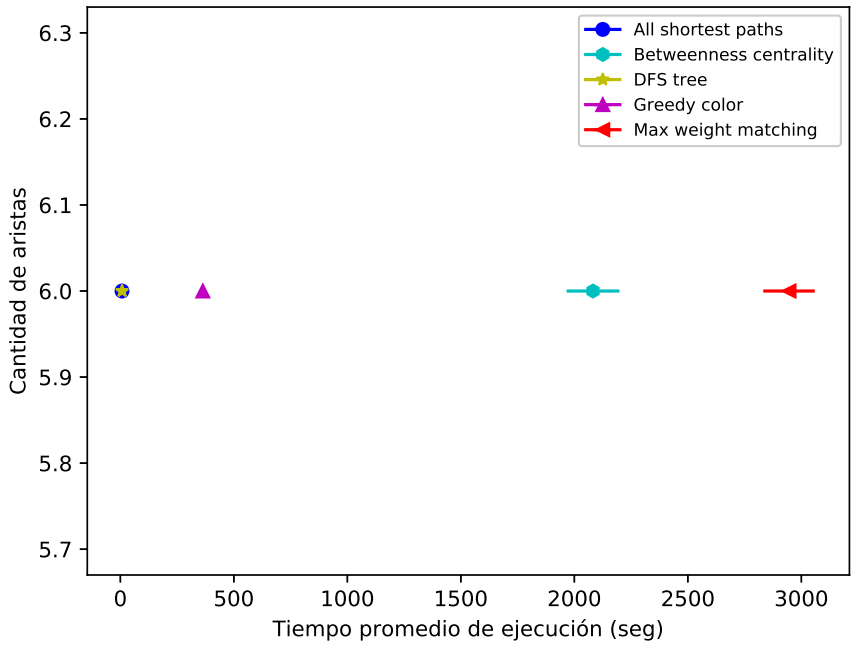

Figura 7: Diagrama de dispersión.

Referencias

[1] Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart *Exploring network structure, dynamics, and function using NetworkX*
`https://networkx.github.io/documentation/stable/`

[2] Schaeffer E. *Optimización de flujo en redes, 2019.*
`https://elisa.dyndns-web.com/teaching/opt/flow/`

[3] Tomihisa Kamada, Satoru Kawai. *An Algorithm for Drawing General Undirected Graphs*
`Information Processing Letters, 1988.`