Tarea 4

5272

1. Introducción

En el presente trabajo se utilizaron tres diferentes tipos de generadores de grafos en este caso los seleccionados fueron: Grafo Estrella, Grafo Paleta, Grafo Rueda; para cada uno se seleccionó una base logarítmica 2 para crear cuatro grafos de orden 4, 8, 16, 32. Los pesos de cada arco se distribuyen normalmente con media 8 y una desviación estándar de 0.3. Se utilizaron además los algoritmos de flujo máximo: Algoritmo Dinitz, Empuje de Preflujo y Ruta de Aumento más Corta.

2. Generadores de Grafos

Los generadores de grafos utilizados se encuentran a continuación:

Grafo Estrella

Este grafo es de orden n, se conoce también como n estrella. Devuelve un grado con $n+1$ como nodo central, conectado a n nodos externos [2].

Grafo Paleta

Este grafo consiste es un subgrafo completo con m vértices y otro subgrafo con n vértices conectados por medio de un puente [1].

Grafo Rueda

Este grafo también es llamado n ruedas, está formado por la conexión de un solo vértice universal a todos los vértices de un ciclo es decir, contiene un ciclo de orden $n-1$ para el cual cada vértice del ciclo esta conectado a otro vértice [6].

3. Algoritmos de Flujo Máximo

Los algoritmos de flujo máximo utilizados se encuentran a continuación:

Algoritmo Dinitz

Es uno de los algoritmos más rápidos y más fácil de implementar, también se le conoce como algoritmo Dinic. Este algoritmo polinomial calcula el flujo máximo, donde regresa una red residual resultante [3].

Empuje de Preflujo

Este tipo de algoritmo maneja un preflujo, es decir, no se satisface necesariamente las condiciones de conservación de flujo, por lo que es posible que un vértice reciba más flujo que el que distribuye. Los algoritmos de preflujo se pueden mejorar significativamente al incorporar condiciones para decidir que nodos elegir, las cuales tienden a ser heurísticas [5].

Ruta de Aumento más Corta

Encuentra un flujo máximo utilizando el algoritmo de ruta de aumento más corta, regresa una red residual resultante de calcular el flujo máximo [4].

4. Resultados Computacionales

Se realiza una evaluación de los datos utilizando herramientas estadísticas como los Boxplot que se describen a continuación. El primer análisis realizado es del tiempo contra los algoritmos de flujo máximo ($Figura1$) donde se puede decir que los algoritmos Dinitz, Preflow Push y Shortest tiene una media de 0.005532, 0.009594 y 0.006333 respectivamente y una desviación de 0.005054, 0.007593 y 0.004624 para cada uno; el segundo Boxplot ($Figura2$) es tiempo contra densidad donde la media está en un rango entre $0{,}06 - 1{,}17$ con tiempos entre $0{,}008680 - 0{,}003434$; para el tercer Boxplot ($Figura3$) la media tiene un comportamiento para cada algoritmo de la siguiente forma: Dinitz: 0.009041, Preflow Push: 0.004764 y Shortest:0.009435 y una desviación Dinitz: 0.008347, Preflow Push: 0.003241 y Shortest: 0.006466; por último se realizó el Boxplot de tiempo contra orden ($Figura4$) quedando de la siguiente forma: para los algoritmos de orden 4 la media y la desviación tienen un valor de 0.002647 y 0.001058 respectivamente, para los algoritmos de orden 8 un valor de 0.004634 para la media y 0.001991 para la desviación, para orden 16 valores de 0.008008 y 0.003711 respectivamente y por último para orden 32 una media de 0.015339 y una desviación de 0.008589.

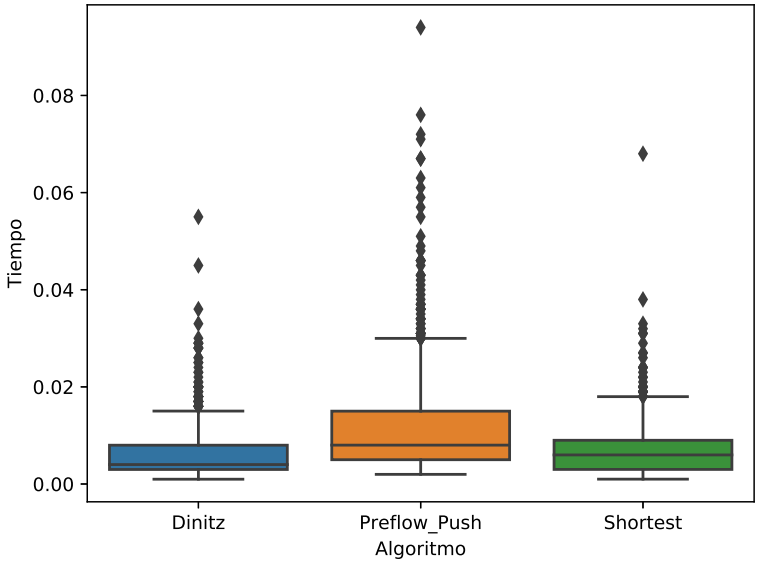

Figura 1: Boxplot Algoritmo

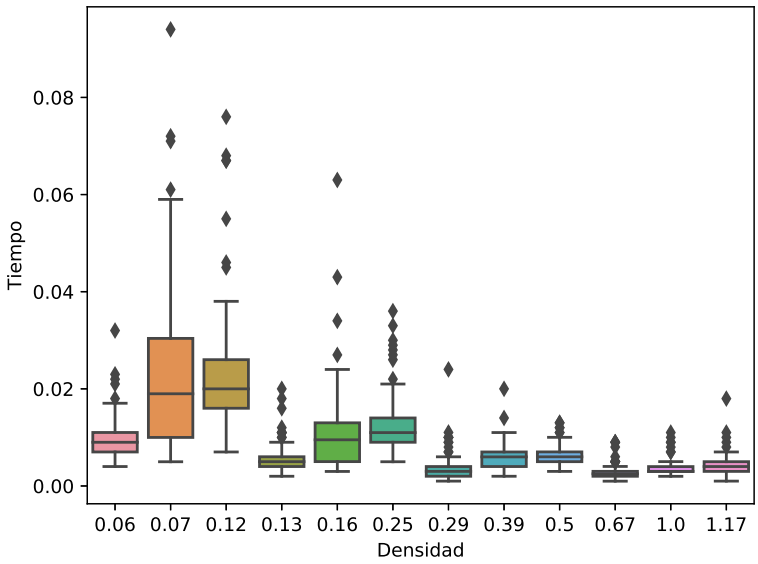

Figura 2: Boxplot Densidad

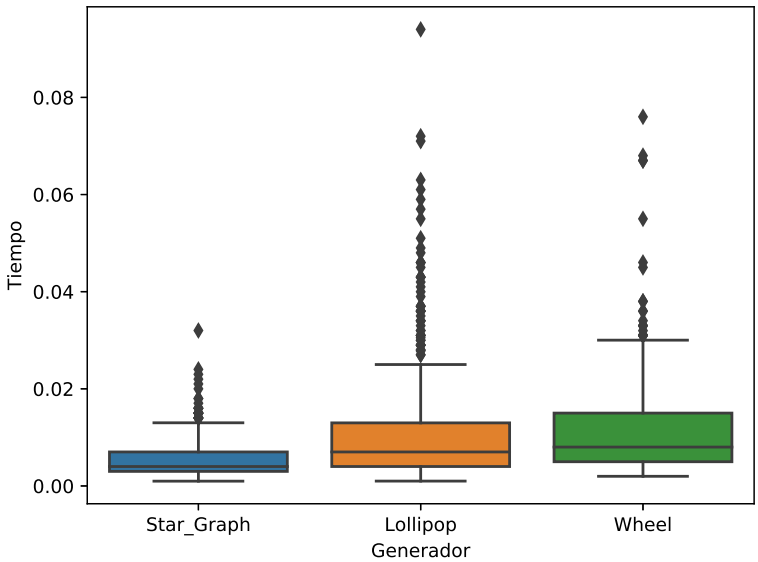

Figura 3: Boxplot Generador

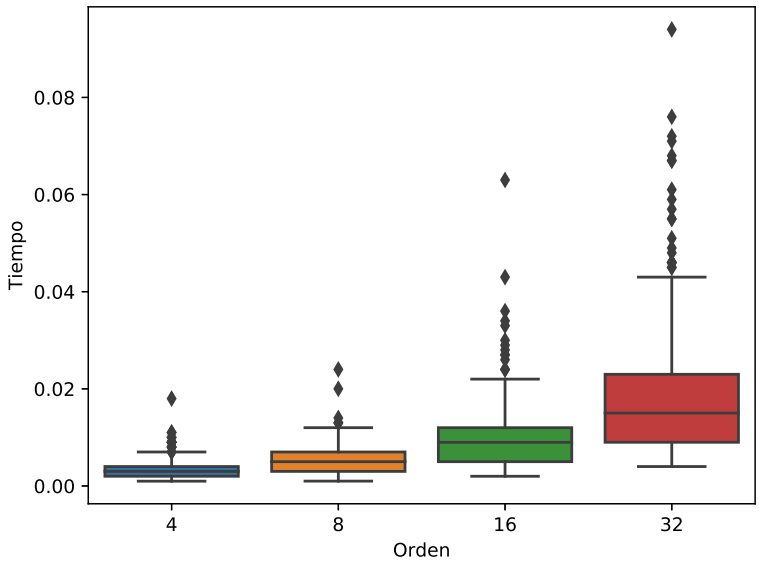

Figura 4: Boxplot Orden

Se realiza además un análisis con un ANOVA para ver si todos los factores estudiados se encuentran relacionados entres sí, lo que se puede decir que como el $valor - p$ es pequeño entonces existe vínculos entre dichos factores.

ANOVA.txt

	sum_sq	df	F	PR>F
Generador	0.001308	2.0	22.024254	3.559232e-10
Algoritmo	0.009094	2.0	153.117628	4.343969e-62
Generador:Algoritmo	0.003071	4.0	25.850057	7.452684e-21
Orden	0.021814	1.0	734.566837	9.845713e-136
Generador:Orden	0.001889	2.0	31.805667	2.678743e-14
Orden:Algoritmo	0.002061	2.0	34.695840	1.651675e-15
Densidad	0.000044	1.0	1.472617	2.250941e-01
Generador:Densidad	0.000003	2.0	0.050462	9.507913e-01
Algoritmo:Densidad	0.000026	2.0	0.445265	6.407255e-01
Orden:Densidad	0.000007	1.0	0.229653	6.318402e-01
Residual	0.052890	1781.0	NaN	NaN

Se realiza una matriz de correlación (*Figura*5) donde se puede comprobar que existe una relación entre factores como se mencionó anteriormente.

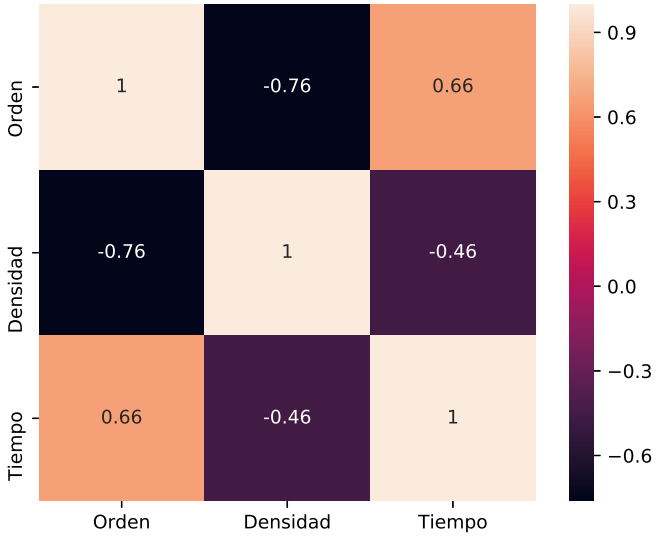

Figura 5: Matriz de Correlación

5. Fragmento de Código

```
1
2  for j in range(10):
3      for i in range(2, 6):
4          i = 2 ** i
5
6          stars_time_G = time()
7          G = nx.star_graph(i)
8          end_time_G = time()
9          total_time_G = end_time_G - stars_time_G
10         total_time_stars.append(total_time_G)
11         ndistribution_weight(8, 0.3, G)
12
13         stars_time_A = time()
14         A = nx.lollipop_graph(3, i)
15         end_time_A = time()
16         total_time_A = end_time_A - stars_time_A
17         total_time_lollipop.append(total_time_A)
18         ndistribution_weight(8, 0.3, A)
19
20         stars_time_D = time()
21         D = nx.wheel_graph(i)
22         end_time_D = time()
23         total_time_D = end_time_D - stars_time_D
24         total_time_wheel.append(total_time_D)
25         ndistribution_weight(8, 0.3, D)
26
27         for r in range(5):
28
29             for k in range(1, 6):
30                 list_random_G = random.sample(range(len(G)), 2)
31                 inicial_time_G = time()
32                 for s in range(1, 6):
33                     value_dinitz = dinitz(G, list_random_G[0], list_random_G[1], capacity="
    weight")
34                 final_time_G = time()
35                 execution_time_G = final_time_G - inicial_time_G
36
37             table1 = pd.DataFrame({'Generador': ['Star_Graph'],
38                                    'Algoritmo': ['Dinitz'],
39                                    'Orden': i,
40                                    'Densidad': round(G.size() / nx.complete_graph(i).size(),2)
    ,
41                                    'Tiempo': execution_time_G + total_time_G})
42             table = table.append(table1)
43
44             for h in range(1, 6):
45                 list_random_G = random.sample(range(len(G)), 2)
46                 inicial_time_G = time()
47                 for s in range(1, 6):
48                     value_preflow_push = preflow_push(G, list_random_G[0], list_random_G[1],
    capacity="weight")
49                 final_time_G = time()
50                 execution_time_G = final_time_G - inicial_time_G
51
52             table4 = pd.DataFrame({'Generador': ['Star_Graph'],
53                                    'Algoritmo': ['Preflow_Push'],
54                                    'Orden': i,
55                                    'Densidad': round(G.size() / nx.complete_graph(i).size
    (),2),
56                                    'Tiempo': execution_time_G + total_time_G})
57             table = table.append(table4)
```

new.py

Referencias

[1] Romain Boulet and Bertrand Jouve. The lollipop graph is determined by its spectrum. *arXiv preprint arXiv:0802.1035*, 2008.

[2] Khaled Day and Anand Tripathi. Unidirectional star graphs. *Information Processing Letters*, 45(3):123–129, 1993.

[3] Shimon Even. *Graph algorithms*. Cambridge University Press, 2011.

[4] Donald B Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM (JACM)*, 24(1):1–13, 1977.

[5] S Kumar and P Gupta. An incremental algorithm for the maximum flow problem. *Journal of Mathematical Modelling and Algorithms*, 2(1):1–16, 2003.

[6] Nasser R Sabar, Masri Ayob, Graham Kendall, and Rong Qu. Roulette wheel graph colouring for solving examination timetabling problems. In *International conference on combinatorial optimization and applications*, pages 463–470. Springer, 2009.