Tarea 4

5280

2 de abril de 2019

Generadores de grafos

En la selección de los algoritmos generadores de grafos de NetworkX [1] se tiene en cuenta la densidad de las aristas que generan.

- `random_tree`: Este algoritmo recibe el número de nodos y devuelve un árbol aleatorio.
- `dense_gnm_random`: Recibe como parámetros directamente el número de nodos y el número de aristas, el algoritmo los distribuye aleatoriamente.
- `erdos_renyi`: Este algoritmo recibe el número de nodos como parámetros y una probabilidad de creación de aristas, por lo tanto la cantidad de aristas varía de un grafo a otro, manteniendo la misma cantidad de nodos.

Algoritmos de flujo máximo

Los algoritmos de flujo máximose eligen de modo que no generen errores cuando se les pasan grafos con nodos no conexos. Los tres reciben como parámetros el grafo, el nodo fuente y el nodo sumidero.

- `edmonds_karp`: Este algoritmos es una implementación del método de Ford-Fulkerson, con la particularidad de que el orden para ir buscando los caminos está definido.
- `dinitz`: La introducción de los conceptos nivel de grafo y bloqueo de flujo es lo que define el rendimiento de este algoritmo.
- `boykov_kolmogorov`: Aunque este algoritmo está pensado para grafos dirigidos, también fuciona con grafos no dirigidos.

Generación de datos

Son generados 10 grafos de 100, 200, 400 y 800 nodos, con cada uno de los tres algoritmos generadores elegidos, empleando pesos con distribición normal alrededor de 11 con una varianza de 7, como puede observarse en la figura 2.

Los datos para realizar el análisis son guardados en una tabla de 1800 filas que consta de las siguientes columnas.

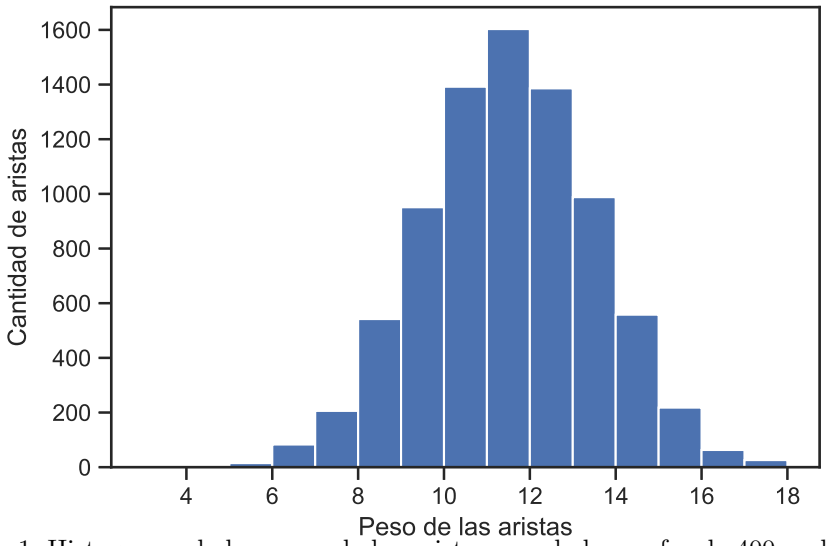

Figura 1: Histograma de los pesos de las aristas uno de los grafos de 400 nodos generados

- Generador: Nombre del algoritmo generador de grafos.
- Algoritmo: Nombre del algoritmo de flujo máximo.
- Nodos: Cantidad de nodos.
- Densidad: Densidad del grafo.
- Mediana: Valor de la mediana obtenida de las cinco iteraciones.
- Media: Valor de la media obtenida de las cinco iteraciones.
- Desv: Valor de la desviación estándar obtenida de las cinco. iteraciones.
- Var: Valor de la varianza obtenida de las cinco iteraciones.

A continuación se muestra el código generador de los datos.

```
def GenerateGraphs(size, base):
    dic=GenerateDic()
    for i in range(1,5):
        for j in range(10):
            G=AddEdges(nx.dense_gnm_random_graph(size,int(size*size*0.03 )))
            nodes=RandNodes(size-1)
            for k in range (5):
                edmond=[]
                din=[]
                boyk=[]
                for l in range(5):
                    edmond.append(Edmond(G,nodes[l][0],nodes[l][1]))
                    din.append(Din(G,nodes[l][0],nodes[l][1]))
                    boyk.append(Boyk(G,nodes[l][0],nodes[l][1]))
                Asign(dic, "dense","Edmond", G.number_of_nodes(),G.number_of_edges()
    ,nodes[k],np.median(edmond),np.mean(edmond),np.std(edmond),np.var(edmond),nx.
    density(G))
                Asign(dic, "dense","Dinitz", G.number_of_nodes(), G.number_of_edges
    (),nodes[k],np.median(din),np.mean(din),np.std(din),np.var(din),nx.density(G))
```

```
                Asign(dic, "dense","Boyk", G.number_of_nodes(), G.number_of_edges(),
    nodes[k],np.median(boyk),np.mean(boyk),np.std(boyk), np.var(boyk),nx.density(G))
            G=AddEdges(nx.erdos_renyi_graph(size, 0.1))
            for k in range (5):
                edmond=[]
                din=[]
                boyk=[]
                for l in range(5):
                    edmond.append(Edmond(G,nodes[l][0],nodes[l][1]))
                    din.append(Din(G,nodes[l][0],nodes[l][1]))
                    boyk.append(Boyk(G,nodes[l][0],nodes[l][1]))
                Asign(dic, "erdos","Edmond", G.number_of_nodes(), G.number_of_edges
    (),nodes[k],np.median(edmond),np.mean(edmond),np.std(edmond),np.var(edmond),nx.
    density(G))
                Asign(dic, "erdos","Dinitz", G.number_of_nodes(), G.number_of_edges
    (),nodes[k],np.median(din),np.mean(din),np.std(din),np.var(din),nx.density(G))
                Asign(dic, "erdos","Boyk", G.number_of_nodes(), G.number_of_edges()
    ,nodes[k],np.median(boyk),np.mean(boyk),np.std(boyk),np.var(boyk),nx.density(G))
            G=AddEdges(AddEdges(nx.random_tree(size)))
            for k in range (5):
                edmond=[]
                din=[]
                boyk=[]
                for l in range(5):
                    edmond.append(Edmond(G,nodes[l][0],nodes[l][1]))
                    din.append(Din(G,nodes[l][0],nodes[l][1]))
                    boyk.append(Boyk(G,nodes[l][0],nodes[l][1]))
                Asign(dic, "tree","Edmond", G.number_of_nodes(), G.number_of_edges()
    ,nodes[k],np.median(edmond),np.mean(edmond),np.std(edmond),np.var(edmond),nx.
    density(G))
                Asign(dic, "tree","Dinitz", G.number_of_nodes(), G.number_of_edges()
    ,nodes[k],np.median(din),np.mean(din),np.std(din),np.var(din),nx.density(G))
                Asign(dic, "tree","Boyk", G.number_of_nodes(), G.number_of_edges(),
    nodes[k],np.median(boyk),np.mean(boyk),np.std(boyk),np.var(boyk),nx.density(G))
        size*=base;
    df=pd.DataFrame(dic)
    df.to_csv("matrix.csv")
```

Generador.py

Análisis de varianza

Para el análisis de varianza, en lo adelante ANOVA, por sus siglas en inglés [2], se utilizó el siguiente código.

```
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt
import researchpy as rp
import statsmodels.api as sm
from statsmodels.formula.api import ols
import numpy as np
import pingouin as pg
import seaborn as sns
from statsmodels.stats.multicomp import pairwise_tukeyhsd
import csv

df = pd.read_csv("Matrix.csv", dtype={'Nodos': 'category', 'Generados': 'category',"
    Densidad": np.float64 })
logX = np.log1p(df["Mediana"])
df = df.assign(mediana=logX.values)
df.drop(["Mediana"], axis= 1, inplace= True)
```

```
17
18 for i in range(0, df["Densidad"].count()):
19     if  df.iat[i, 9]  < 0.03716667:
20         df.iat[i, 9] = 1
21     elif df.iat[i, 9]  < 0.07183333:
22         df.iat[i, 9] = 2
23     else:
24         df.iat[i, 9] = 3
25 df["Densidad"].replace({1: 'baja', 2: 'media', 3: 'alta'}, inplace= True)
26 factores=["Nodos","Generador","Algoritmo","Densidad"]
27 for i in factores:
28     print(rp.summary_cont(df['mediana'].groupby(df[i])))
29     anova = pg.anova (dv='mediana', between=i, data=df, detailed=True , )
30     pg._export_table (anova,("ANOVA"+i+".csv"))
31     ax=sns.boxplot(x=df["mediana"], y=df[i], data=df, palette="Set1")
32     plt.savefig(i+"1.png", bbox_inches='tight')
33     plt.savefig( i + "1.eps", bbox_inches='tight')
34     tukey = pairwise_tukeyhsd(endog = df["mediana"], groups= df[i], alpha=0.05)
35     tukey.plot_simultaneous(xlabel='Time', ylabel=i)
36     plt.vlines(x=49.57,ymin=-0.5,ymax=4.5, color="red")
37     plt.savefig(i+"2.png", bbox_inches='tight')
38     plt.savefig(i + "2.eps", bbox_inches='tight')
39     t_csv = open("Tukey"+i+".csv", 'w')
40     with t_csv:
41         writer = csv.writer(t_csv)
42         writer.writerows(tukey.summary())
43     plt.show()
```

Anova.py

Efecto que el generador de grafo usado tiene en el tiempo de ejecución

El ANOVA realizado con los datos obtenidos sobre las mediciones arroja los resultados mostrados en el cuadro 1, o sea, que el generador de grafos sí influye en el tiempo de los algoritmos de flujo.

Cuadro 1

Grupo 1	Grupo 2	Grupo 3	Menos	Mayor	Rechazar
dense	erdos	0.0971	0.0538	0.1405	True
dense	tree	-0.1993	-0.2427	-0.156	True
erdos	tree	-0.2964	-0.3398	-0.2531	True

La figura 1 muestra el diagrama de caja y bigotes para estas mediciones, donde se aprecia que el generador `dense_gnm_random` es el que más influye en el tiempo de ejecución.

Efecto que el algoritmo de flujo máximo usado tiene en el tiempo de ejecución

El cuadro 2 muestra el resultado ANOVA sobre los algoritmos de flujo máximo. Puede observarse que al comparar `boykov_kolmogorov` y `edmonds_karp`, no existen diferencias entre ellos en cuanto a

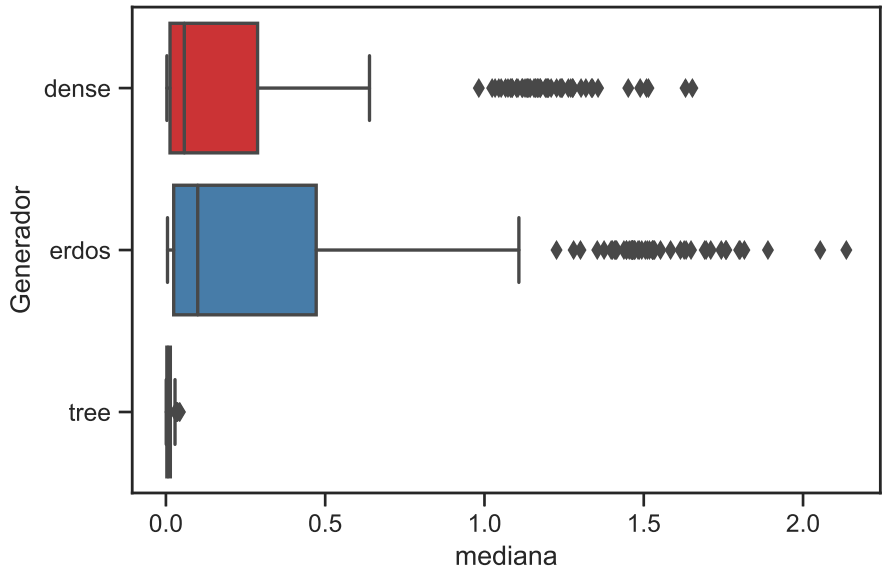

Figura 2: Diagrama de caja y bigotes para los generadores de grafo.

cómo influencian en el tiempo de ejecución, mientras `dinitz` sí tiene un comportamiento diferente a ellos. Esto se observa con mayor claridad en el diagrama de caja y bigotes de la figura 6.

Cuadro 2

Grupo 1	Grupo 2	Grupo 3	Menos	Mayor	Rechazar
Boyk	Dinitz	0.1893	0.1448	0.2338	True
Boyk	Edmond	-0.034	-0.0786	0.0105	False
Dinitz	Edmond	-0.2234	-0.2679	-0.1789	True

Efecto que el número de nodos del grafo tiene en el tiempo de ejecución

En el cuadro 3 se observa que entre grafos de 100 y 200 nodos no hay un efecto significativo en el tiempo de ejecución. A madida que aumenta el tamaño el efecto va incrementándose. En la figura 3 puede observarse esto gráficamente en el diagrama de caja y bigotes.

Efecto que la densidad del grafo tiene en el tiempo de ejecución

Para realizar el análisis del efecto de la densidad se realiza un análisis de la distribución de los valores, pues el número de valores de densidad es excesivo para analizarlos en su totalidad. En la figura 7 se muestra el histograma realizado sobre las 1800 mediciones realizadas y puede verse que están distribuidos uniformemente en tres grupos de densidad: baja, media y alta.

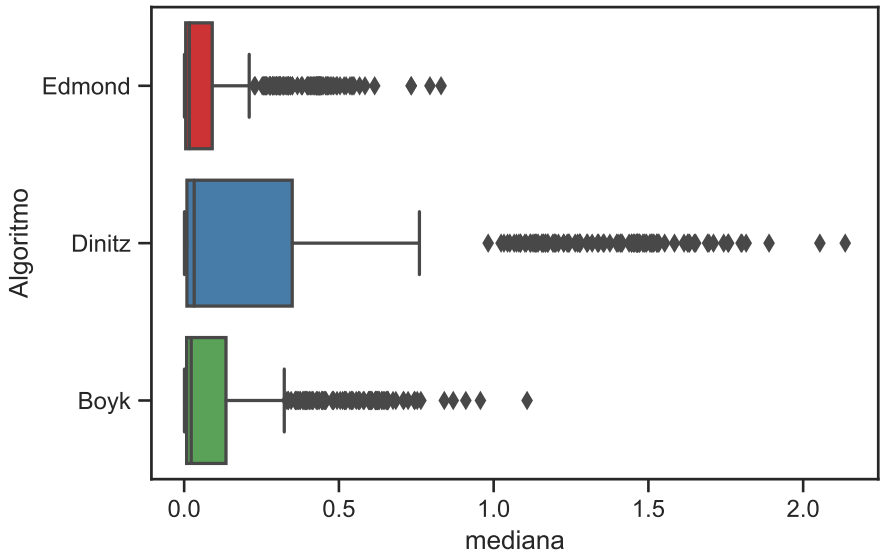

Figura 3: Diagrama de caja y bigotes para los algoritmos de flujo máximo.

Cuadro 3

Grupo 1	Grupo 2	Grupo 3	Menos	Mayor	Rechazar
100	200	0.0216	-0.0252	0.0685	False
100	400	0.1322	0.0853	0.179	True
100	800	0.5155	0.4686	0.5623	True
200	400	0.1105	0.0636	0.1574	True
200	800	0.4938	0.447	0.5407	True
400	800	0.3833	0.3364	0.4302	True

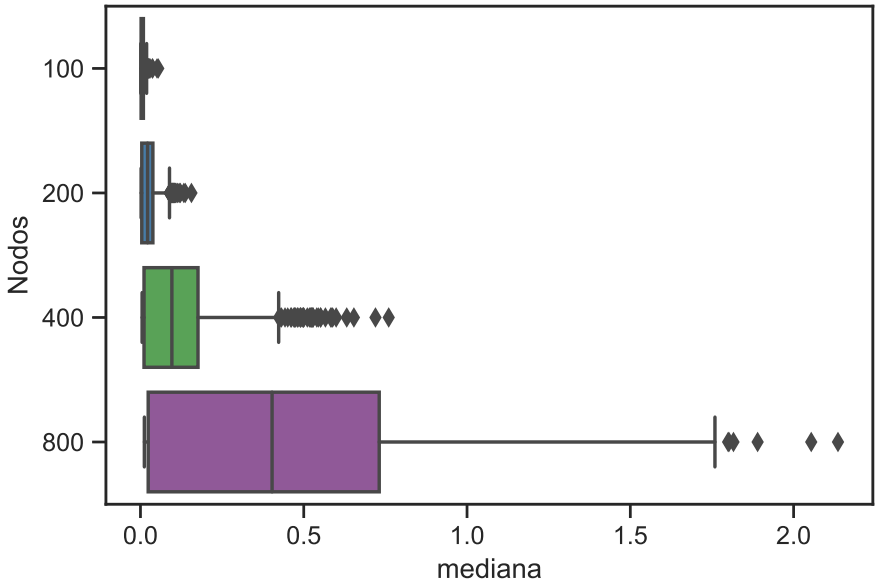

Figura 4: Diagrama de caja y bigotes para el número de nodos de los grafos.

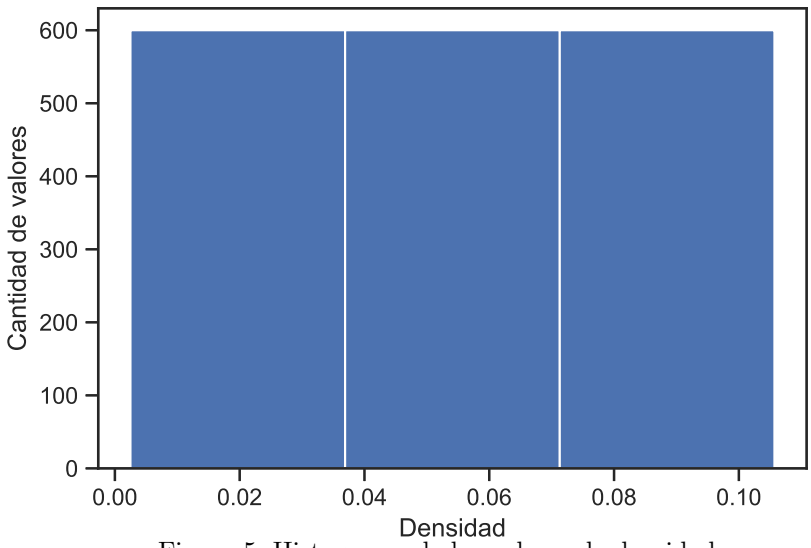

Figura 5: Histograma de los valores de densidad.

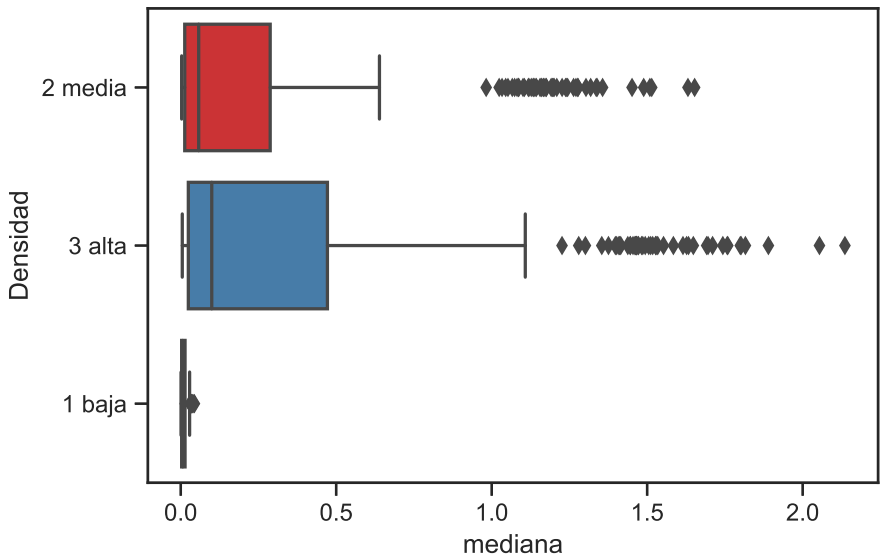

Figura 6: Diagrama de caja y bigotes para la densidad de los grafos.

En el cuadro 4 se muestran los resultados del ANOVA para los valores de densidad y se puede apreciar que sí tiene efecto en el tiempo de ejecución de los algoritmos. Esto se observa de manera gráfica en la figura 8.

Cuadro 4

Grupo 1	Grupo 2	Grupo 3	Menos	Mayor	Rechazar
baja	media	0.1993	0.156	0.2427	True
baja	alta	0.2964	0.2531	0.3398	True
media	alta	0.0971	0.0538	0.1405	True

Conclusiones

Al realizar un ANOVA sobre los cuatro parámetros, puede apreciarse en el cuadro 5 que la mayor influencia está dada entre los algoritmos de flujo máximo y el número de nodos de los grafos.

Cuadro 5

	sum_sq	df	F	PR(¿F)
Generador	-0.009385	2	-0.38830974	1
Algoritmo	-0.08859016	2	-3.66547001	1
Densidad	-0.00501818	2	-0.20762995	1
Generador:Algoritmo	0.03866208	4	0.79983312	0.37126377
Generador:Densidad	0.03866208	4	0.79983323	0.37126374
Algoritmo:Densidad	0.01865205	4	0.38586979	0.67991571
Nodos	1.03E-11	1	8.53E-10	0.99997669
Algoritmo:Nodos	23.596196	2	976.306468	3.16E-287
Nodos:Densidad	0.00972681	2	0.40245243	0.66873877
Generador:Nodos	0.00915802	2	0.37891829	0.68465657
Residual	21.5827752	1786		

Referencias

[1] Networkx. https://networkx.github.io/documentation/stable/reference/algorithms/.

[2] Python for data science. https://pythonfordatascience.org/anova-python/.