Tarea 4 Optimización de Flujo en Redes

1985274

2 de abril de 2019

En este trabajo se presenta un análisis en el que se han seleccionado 3 algoritmos generadores de grafos y a su vez se han escogido 3 implementaciones de flujo máximo. Para esto se ha utilizado el lenguaje **Python** en su versión 3.7 [11], el editor de código Spyder en su versión 3.3.1 y el editor Texmaker 5.0.2 para redactar el documento. Se utilizan además las librerías **networkX** 1.5 [8], **Matplotlib** [6], **Numpy** [9], la librería **Panda** [10], la librería **Scipy** [13], la librería **Seaborn** [7] para graficar, la librería **Researchcpy** [2] para producir pandas Dataframe de pruebas estadísticas y **Pingouin** [12] para realizar el análisis de varianza de un factor (ANOVA).

1. Generadores

Se utilizaron los generadores siguientes: *powerlaw cluster graph*, *watts strogatz graph*, *connected watts strogatz graph* y como algoritmos de flujo máximo los siguientes algoritmos: *maximun flow value*, *dinitz* y *shortest augmenting path*.

- Algoritmo *powerlaw cluster graph*: o distribución en grados y agrupamiento promedio aproximado. Recibe 4 parámetros. Este algoritmo a cada borde aleatorio va seguido de una posibilidad haciendo una ventaja a uno de sus vecinos también.
- Algoritmo *watts strogatz graph*: Este es un algoritmo recibe cuatro parámetros. Primero crea un anillo sobre n nodos y luego cada nodo en el anillo es conectado con sus vecinos más cercanos. Luego se crean accesos directos reemplazando algunos bordes.
- Algoritmo *connected watts strogatz graph*: recibe 4 parámetros que son la cantidad de anillos, la cantidad de vecinos más cercano en topología anillo, la probabilidad de volver a cablear ese borde y la semilla para generar números aleatorios.

2. Algorítmos de flujo máximo

- Algoritmo *maximum flow value*: recibe varios parámetros entre ellos el grafo, el nodo origen para el flujo, el nodo destino. En este algoritmo el objetivo es encontrar el costo de valor máximo de un solo producto.

- Algoritmo *dinitz*: esta función devuelve el residual resultante después de calcular el flujo máximo.
- Algoritmo *shortest augmenting path* o ruta de aumento más corta. Este algoritmo tiene como idea central que cada ruta de aumento más corta se encuentra en tiempo O(m), y cuando se aumenta el flujo a lo largo de él, al menos un borde está saturado(el flujo alcanza la capacidad). Cada vez que un borde se satura, la distancia desde la fuente debe aumentar y esta distancia es a lo sumo O(n) porque cada borde puede estar saturado a lo sumo O(n) veces [4].

A continuación un fragmento del Dataset de salida que tiene todas las corridas.

grafo	generador	algoritmo_flujo	vertices	aristas	fuente	sumidero	media	mediana	varianza	desviacion
vertices119aristas847	powerlaw_cluster_graph	shortest_augmenting_path	119	847	58	87	0.0076	0.00658	5.00E-05	0.00704
vertices119aristas847	powerlaw_cluster_graph	dinitz	119	847	58	87	0.02944	0.03124	1.00E-05	0.00364
vertices119aristas847	powerlaw_cluster_graph	maximum_flow	119	847	58	87	0.01068	0.01554	8.00E-05	0.00905
vertices119aristas847	powerlaw_cluster_graph	shortest_augmenting_path	119	847	58	87	0.00939	0.01558	6.00E-05	0.00766
vertices119aristas847	powerlaw_cluster_graph	dinitz	119	847	58	87	0.03387	0.03129	1.00E-05	0.0032
vertices119aristas847	powerlaw_cluster_graph	maximum_flow	119	847	58	87	0.01259	0.01563	5.00E-05	0.0073
vertices119aristas847	powerlaw_cluster_graph	shortest_augmenting_path	119	847	58	87	0.00806	0.00899	5.00E-05	0.00701
vertices119aristas847	powerlaw_cluster_graph	dinitz	119	847	58	87	0.03326	0.03484	0.0001	0.01022
vertices119aristas847	powerlaw_cluster_graph	maximum_flow	119	847	58	87	0.0125	0.01562	4.00E-05	0.00625
vertices119aristas847	powerlaw_cluster_graph	shortest_augmenting_path	119	847	58	87	0.00626	0	6.00E-05	0.00767

Cuadro 1: Fragmento del Dataset

2.1. Código

```
from networkx.algorithms.flow import maximum_flow_value
from networkx.algorithms.flow import shortest_augmenting_path
import datetime as dt
import pandas as pd
import statistics as stats

mu, sigma = 15, 0.2
cantidad_instancias_grafo = 10
rango_instancias_grafo = 11
mediciones = 5
tipo_x_nodos = 4
base_calculo_nodos = 2.6
archivo_CSV = "Datoss.csv"
base_inicio_calculo_nodos = 4
probabilidad = 0.215
control_iteraciones = 0
generadores_grafos = {"powerlaw_cluster_graph":nx.powerlaw_cluster_graph,
    "watts_strogatz_graph": nx.watts_strogatz_graph, "
    connected_watts_strogatz_graph": nx.connected_watts_strogatz_graph }
algoritmos_flujo = { "maximum_flow_value": maximum_flow_value,  "dinitz":
     dinitz,"shortest_augmenting_path": shortest_augmenting_path}
```

```
estructura_CSV = {"grafo": [],"generador": [], "algoritmo_flujo": [], "vertices": [],"aristas": [],"fuente": [],"sumidero": [],"densidad": [],"media": [],"mediana": [],
    "varianza": [],"desviacion": []}
for generador_grafo in generadores_grafos:
    for instancia_grafo_x_nodos in [round(pow(base_calculo_nodos, value + 1))
                                    for value in
                                    range(base_inicio_calculo_nodos, base_inicio_calculo_nodos + tipo_x_nodos)]:
        for grafo in range(1, cantidad_instancias_grafo + 1):
            USGraph = generadores_grafos[generador_grafo](instancia_grafo_x_nodos, round((instancia_grafo_x_nodos * probabilidad) / 2),probabilidad, seed=None)
            fuente = np.random.randint(1, high=(instancia_grafo_x_nodos - 1), dtype="int")
            sumidero = np.random.randint(1, high=(instancia_grafo_x_nodos - 1), dtype="int")
            while sumidero == fuente:
                fuente = np.random.randint(1, high=(instancia_grafo_x_nodos - 1), dtype="int")
                sumidero = np.random.randint(1, high=(instancia_grafo_x_nodos - 1), dtype="int")
            if fuente > sumidero:
                swapping = fuente
                fuente = sumidero
                sumidero = swapping
            aristas = USGraph.number_of_edges()
            pesos_normalmente_distribuidos = np.random.normal(mu, sigma, aristas)
            loop = 0
            for (u, v) in USGraph.edges():
                USGraph.edges[u, v]["capacity"] = pesos_normalmente_distribuidos[loop]
                loop += 1
            for instancia_grafo in range(1, 11):
                for algoritmo_flujo in algoritmos_flujo:
                    matriz_tiempos_ejecucion = []
                    for medicion in range(1, mediciones + 1):
                        hora_inicio = dt.datetime.now()
                        obj = algoritmos_flujo[algoritmo_flujo](USGraph, fuente, sumidero, capacity="capacity")
                        hora_fin = dt.datetime.now()
                        tiempo_consumido_segundos = (hora_fin - hora_inicio).total_seconds()
                        matriz_tiempos_ejecucion.append(tiempo_consumido_segundos)
                    media = stats.mean(matriz_tiempos_ejecucion)
                    if media == 0:
```

```
53                    print("iteracion %s tiempo consumido promedio %s" %
    (control_iteraciones + 1, round(media, 4)))
54                    control_iteraciones +=1
55                    estructura_CSV["grafo"].append("vertices" + str(
    instancia_grafo_x_nodos) + "aristas" + str(aristas))
56                    estructura_CSV["algoritmo_flujo"].append(
    algoritmo_flujo)
57                    estructura_CSV["generador"].append(generador_grafo)
58                    estructura_CSV["vertices"].append(
    instancia_grafo_x_nodos)
59                    estructura_CSV["aristas"].append(aristas)
60                    estructura_CSV["fuente"].append(fuente)
61                    estructura_CSV["sumidero"].append(sumidero)
62                    estructura_CSV["densidad"].append(round(nx.density(
    USGraph),5))
63                    estructura_CSV["media"].append(round(media, 5))
64                    estructura_CSV["mediana"].append(round(stats.median(
    matriz_tiempos_ejecucion), 5))
65                    estructura_CSV["varianza"].append(round(stats.
    pvariance(matriz_tiempos_ejecucion, mu=media), 5))
66                    estructura_CSV["desviacion"].append(round(stats.
    pstdev(matriz_tiempos_ejecucion, mu=media), 5))
67                    matriz_tiempos_ejecucion = []
```

3. Análisis de varianza ANOVA

Luego se realizó un análisis de varianza (ANOVA) de un factor sobre estos datos, con el objetivo de comparar los tratamientos en cuanto a sus medias poblacionales. Este análisis de varianza (ANOVA) permite analizar a partir de una hipótesis si el factor tiene o no impacto en la variable tiempo [3].

La estimación se realizó mediante el modelo siguiente.

$$yij = \mu 1 + ti + \varepsilon ij \tag{1}$$

en donde:

$$\mu 1: \quad \text{representa la media de la población} \tag{2}$$

$$ti: \quad \text{representa el efecto del tratamiento i} \tag{3}$$

$$\varepsilon ij: \quad \text{representa el error con una distribución normal} \tag{4}$$

A continuación se obtuvieron las tablas con los resultados de la prueba de ANOVA que son las siguientes:

Cuadro 2: de Generadores

Source	SS	DF	MS	F	p value	np2
Generador	0.653	2	0.326	0.252	0.777	0
within	2327	1797	1.295	-	-	-

Cuadro 3: de Algoritmos de Flujo

Source	SS	DF	MS	F	p value	np2
Algoritmo flujo	553.471	2	276.735	280.187	1.21e-106	0.238
within	1774	1797	0.988	-	-	-

Cuadro 4: de Vértices

Source	SS	DF	MS	F	p value	np2
Vértices	1211.972	3	403.991	649.938	5.13e-286	0.521
within	1116.634	1796	0.622	-	-	-

Cuadro 5: de Aristas

Source	SS	DF	MS	F	p value	np2
Aristas	1214.348	43	28.241	44.516	3.27e-246	0.522
within	1113.988	1756	0.634	-	-	-

4. Prueba estadística

En esta sección se presentan los resultados de la prueba estadística *Tukey* o Método de Tukey (*Honestly significant difference*). Este método se utiliza luego de aplicar el ANOVA y está basada en una prueba de rango estudiantil. Una prueba de ANOVA arroja si los resultados son significativos en general, pero no aclara dónde se encuentran esas diferencias. Se emplean métodos como Tukey para averiguar qué medios de grupos específicos son diferentes, o sea para crear intervalos de confianza para todas las diferencias en parejas entre las medias de los niveles de los factores y además controla la tasa de error por familia en un nivel especificado [5].

En esta prueba se construyen intervalos de confianza para todas las posibles comparaciones por parejas que sigue la forma:

$$(\bar{y_1} - \mu_1), (\bar{y_2} - \mu_2), ...(\bar{y_I} - \mu_I) \tag{5}$$

Este método de *Tukey* resuelve el contraste:

$$H_0 : \mu_i = \mu_j \quad \text{vs } H_1 : \mu_i \neq \mu_j \tag{6}$$

$$H_0 : \quad \text{Hipótesis a demostrar} \tag{7}$$

$$H_1 : \quad \text{Hipótesis alternativa} \tag{8}$$

Este contraste es la hipótesis que se demostrará si es verdadera o no.
A continuación se presentan los resultados del método de *Tukey*.

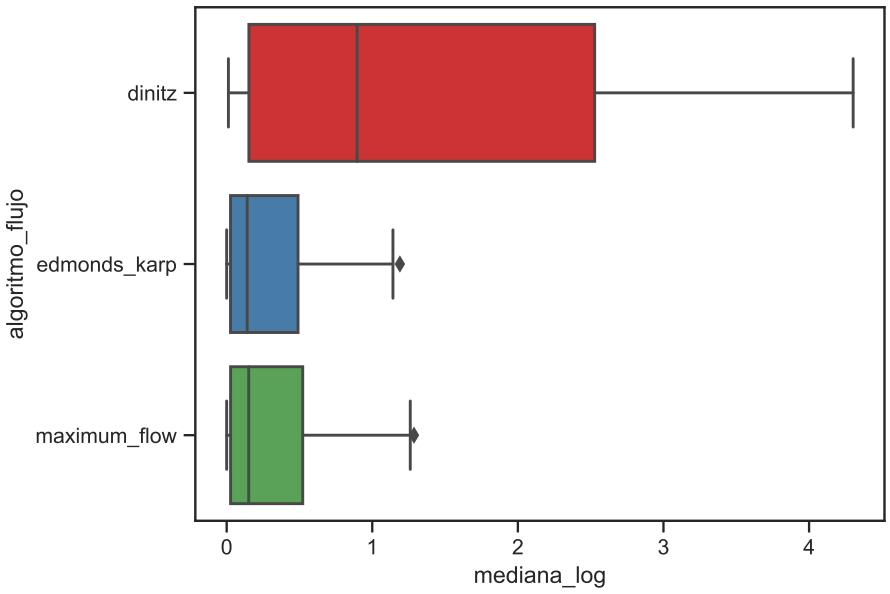

Figura 1: Diagrama de caja de algoritmo de flujo y la mediana

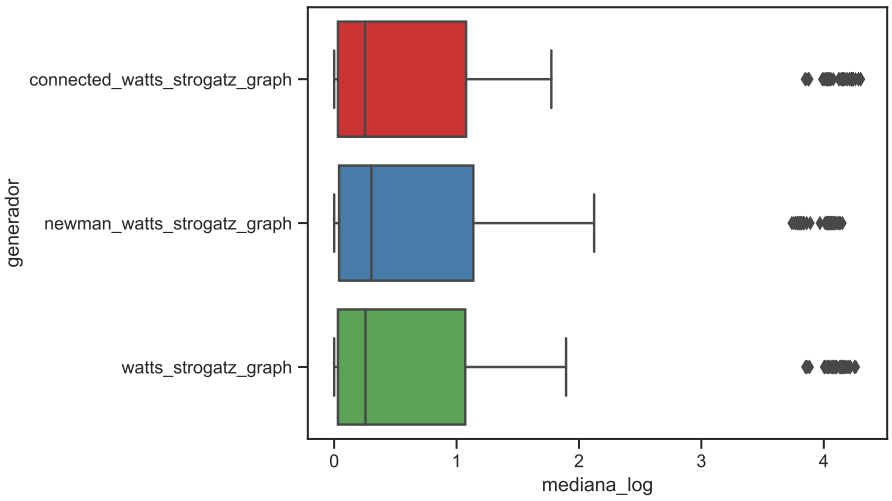

Figura 2: Diagrama de caja de generadores y la mediana

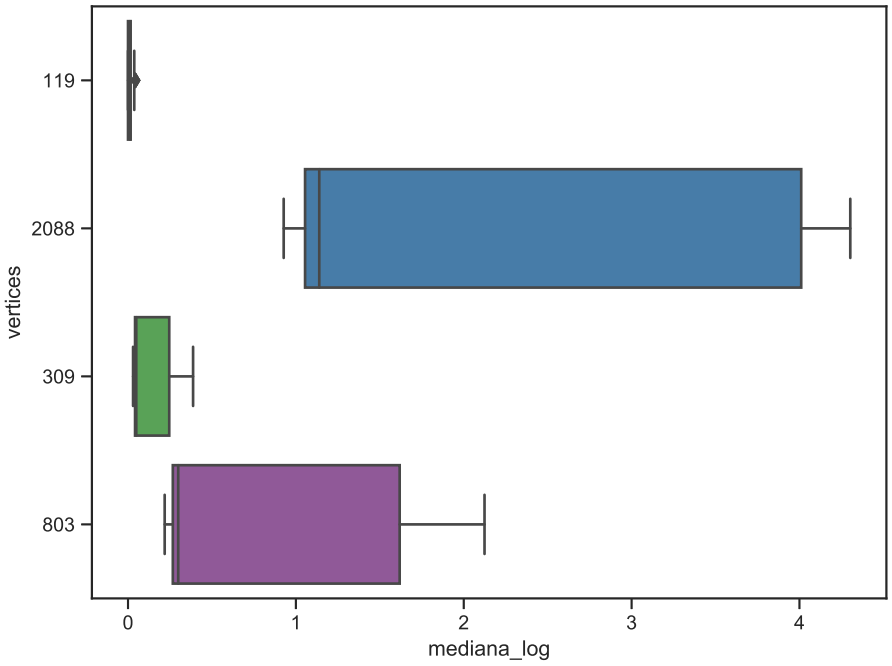

Figura 3: Diagrama de caja de vértices y la mediana

A continuación las gráficas de bigote del método *Tukey*.

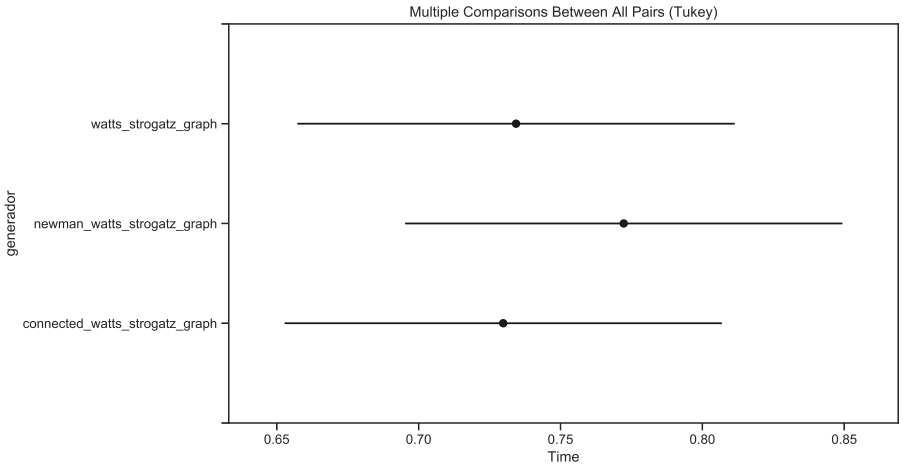

Figura 4: Gráfica de generadores y el tiempo

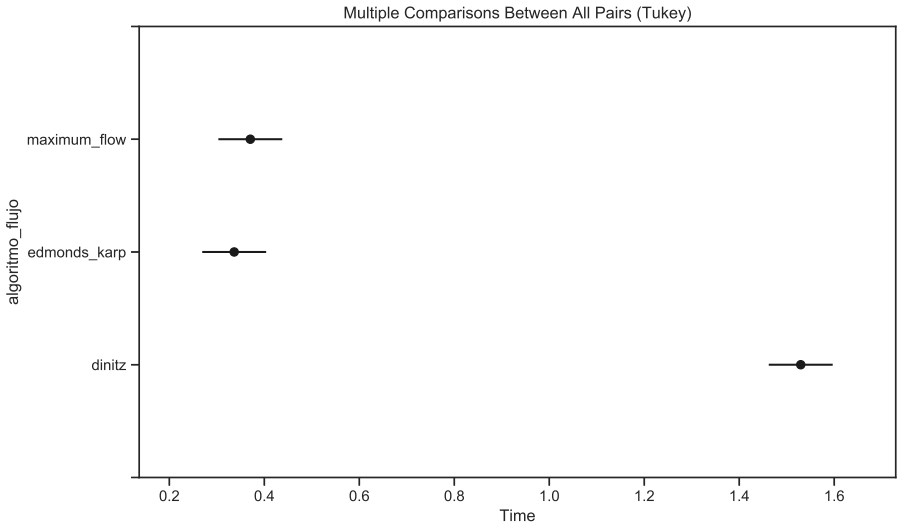

Figura 5: Gráfica de algoritmos de flujo y el tiempo

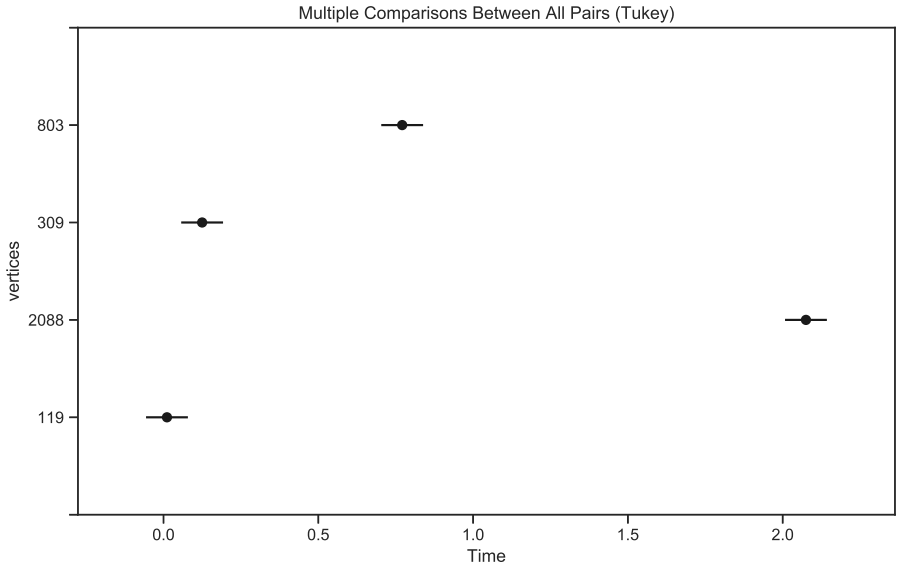

Figura 6: Gráfica de vértices y el tiempo

A continuación le sigue el código para aplicar la prueba de ANOVA y luego la prueba estadística de *Tukey*. La variable *tukey* almacena los resultados de la prueba Tukey.

4.1. Código

```
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("Datos.csv", index_col=None,usecols=[1,2,3,4,8],dtype={'
    generador': 'category',
                                                                    '
    algoritmo_flujo': 'category','vertices': 'category','aristas': '
    category', 'mediana': np.float64} )
logX = np.log1p(df['mediana'])
df = df.assign(mediana_log=logX.values)
df.drop(['mediana'], axis= 1, inplace= True)
factores=["vertices","generador","aristas","algoritmo_flujo"]
```

```
plt.figure(figsize=(8, 6))
for i in factores:
    print(rp.summary_cont(df['mediana_log'].groupby(df[i])))
    anova = pg.anova (dv='mediana_log', between=i, data=df, detailed=True
    )
    pg._export_table (anova,("ANOVA"+i+".csv"))
    ax=sns.boxplot(x=df["mediana_log"], y=df[i], data=df, palette="Set1")
    plt.savefig("boxplot_"+ i+".png", bbox_inches='tight')
    plt.savefig("boxplot_" + i + ".eps", bbox_inches='tight')
    tukey = pairwise_tukeyhsd(endog = df["mediana_log"], groups= df[i],
    alpha=0.05)
    tukey.plot_simultaneous(xlabel='Time', ylabel=i)
    plt.vlines(x=49.57,ymin=-0.5,ymax=4.5, color="red")
    plt.savefig("simultaneous_tukey"+ i+".png", bbox_inches='tight')
    plt.savefig("simultaneous_tukey" + i + ".eps", bbox_inches='tight')
    print(tukey.summary())
    t_csv = open("Tukey"+i+".csv", 'w')
    with t_csv:
        writer = csv.writer(t_csv)
        writer.writerows(tukey.summary())
    plt.show()
```

5. Conclusión

Se puede concluir luego del análisis del comportamiento de los datos reflejados en las tablas y en las gráficas del método estadístico *Tukey* que de los factores que son los Generadores, la cantidad de nodos, la cantidad de aristas y el algoritmo de flujo máximo, el factor que más influye en la variable dependiente (el tiempo) es el factor Generador, luego le sigue el factor cantidad de nodos, y luego otro factor que influye es el algoritmo de flujo máximo y por tanto se puede llega a la conclusión de que sí inlfuyen en el tiempo.

Referencias

[1] Aric Hagberg. `https://www.csie.ntu.edu.tw/~azarc/sna/networkx/networkx/generators/random_graphs.py/`.

[2] Corey Bryant. `https://pypi.org/project/researchpy/`.

[3] Jorge Fallas. *Análisis de varianza*. 2012.

[4] Magnus Lie Hetland. *Python Algorithms: Mastering Basic Algorithms in the Python Language*. 2012.

[5] Lara Porras. `http://wpd.ugr.es/~bioestad/wp-content/uploads/ComparacionesMultiples.pdf/`.

[6] Matplotlib developers. https://matplotlib.org.

[7] Michael Waskom. https://seaborn.pydata.org/.

[8] Networkx developers con última actualización el 19 de Septiembre 2018. https://networkx.github.io/documentation/stable.

[9] Numpy developers. https://www.numpy.org/.

[10] Olivier Pomel. https://pandas.pydata.org/.

[11] Python Software Foundation Versión. https://www.python.org.

[12] Raphael Vallat. https://pypi.org/project/pingouin/.

[13] Travis Oliphant. https://www.scipy.org/.