Tarea No.4: Flujo en Redes

Dayli Machado (5275)

2 de abril de 2019

1. Objetivo

Determinar mediante un diseño de experimentos empleando el análisis de varianzas de un factor y otras pruebas estadísticas la influencia que puede tener en la variable dependiente *tiempo de ejecución*, el orden y la densidad del grafo así como el generador de grafos y el algoritmo de flujo máximo seleccionado.

2. Generador de grafos, algoritmos de flujo máximo empleados y generación del .csv

De los tipos de generador de grafos se seleccionaron los generadores aleatorios y dentro de estos el grupo de tres modelos de generadores de grafos desarrollados por ~~los autores~~ *Watts y Strogatz*[5]. Estos permiten de una forma relativamente sencilla y con menor número de parámetros generar los grafos. Una propiedad importante de estos tres generadores de grafos es que se desarrollan bajo la teoría de red de mundos pequeños, bajo esta teoría se crean nodos principales los cuales están alejados entre sí y generalmente se grafican más grandes que los demás, alrededor de estos se crean nodos que sí son vecinos entre sí y se grafican con tamaños más pequeños, de esta manera se garantiza la propiedad de crear grupos dentro del mismo grafo permitiendo que sea relativamente fácil realizar la visita entre todos los nodos. Esta propiedad refleja mejor el comportamiento de fenómenos reales como las redes eléctricas, neuronales y sociales. Otra propiedad de estos generadores de grafos es que la distancia esperada entre dos nodos elegidos al azar crece de manera proporcional al logaritmo de la cantidad de nodos de la red mientras no se trate de los nodos que están más agrupados, propiedad que detectaron los creadores a partir del comportamiento real de diferentes fenómenos[2].De ahí que los generadores de grafos seleccionados fueron los siguientes:

- *grafo de watts strogatz newman*
- *grafo de watts strogatz*
- *grafo de watts strogatz conectado*

Los algoritmos de flujo máximo seleccionados fueron los siguientes:

- *Algoritmo Boykov-Kolmogorov*
- *Algoritmo de flujo máximo*
- *Algoritmo Edmonds-Karp*

El algoritmo de máximo flujo determina la ruta a través de la cual puede pasar el máximo flujo, de ahí que uno de los parámetros que requiere es la capacidad, y un su defecto la asume como infinita [6]. Se recomienda emplearlo en grafos dirigidos pero funciona también para no dirigidos.

El algoritmo de *Boykov-Kolmogorov* encuentra el flujo máximo de un sólo producto este devuelve la red residual resultante después de calcular el flujo máximo, debe tener capacidad en sus pesos sino los toma como infinitos [3]. Se recomienda para grafos dirigidos, aunque puede emplearse también para no dirigidos.

El algoritmo de *Edmonds-Karp* calcula el flujo máximo de un producto y además devuelve la red residual del mismo, debe emplearse con grafos dirigidos aunque también se emplea para no dirigidos. Al igual que los anteriores debe asignarse una capacidad sino la toma como infinita [4].

A continuación se muestra el fragmento de código desarrollado para generar los grafos empleando los diferentes algoritmos:

```
genera_grafo = {"newman_watts_strogatz_graph": nx.newman_watts_strogatz_graph,
                "watts_strogatz_graph": nx.watts_strogatz_graph,
                "connected_watts_strogatz_graph": nx.
    connected_watts_strogatz_graph}

algoritmos_flujomax = { "maximum_flow": nx.maximum_flow,
                       "boykov_kolmogorov": boykov_kolmogorov,
                       "edmonds_karp": edmonds_karp}
```

Tarea4csvfinal.py

```
for generador_grafo in genera_grafo:
    for instancia_grafo_x_nodos in [round(pow(2.6, value + 1)) for value in range(4,
     8)]: # eleva a la potencia partiendo de la base 2.6
```

Tarea4csvfinal.py

En el siguiente fragmento se muestra como se ha desarrollado el código para asignar el peso normalmente distribuido a los ejes, apoyándose en [1].

```
            grafo_temp = genera_grafo[generador_grafo](instancia_grafo_x_nodos,
                                                       round((
    instancia_grafo_x_nodos * 0.15) / 2),
                                                       0.15,
                                                       seed=None)

            aristas = grafo_temp.number_of_edges()
            pesos_normalmente_distribuidos = np.random.normal(15, 0.2, aristas)

            increment = 0 # incremento para que itere dentro del for que es el grafo
    , magia!!!
            for (u, v) in grafo_temp.edges():
                grafo_temp.edges[u, v]["capacity"] = pesos_normalmente_distribuidos[
    increment]
                increment += 1

            for instancia_grafo in range(1, 6):
                for algoritmo_flujo in algoritmos_flujomax:
                    tabla_tiempo_ejec = []
                    for medicion in range(1, 6):
                        hora_inicio = dt.datetime.now()
                        obj = algoritmos_flujomax[algoritmo_flujo](grafo_temp,
    fuente, sumidero, capacity="capacity")
                        hora_fin = dt.datetime.now()
                        tiempo_consumido_segundos = (hora_fin - hora_inicio).
    total_seconds()
                        tabla_tiempo_ejec.append(tiempo_consumido_segundos)
                    media = stats.mean(tabla_tiempo_ejec)
```

Tarea4csvfinal.py

Después se genera el .csv con la siguiente estructura:

```
                    estructura_CSV["grafo"].append("vertices" + str(
    instancia_grafo_x_nodos) + "aristas" + str(aristas))
                    estructura_CSV["algoritmo_flujo"].append(algoritmo_flujo)
                    estructura_CSV["generador"].append(generador_grafo)
                    estructura_CSV["vertices"].append(instancia_grafo_x_nodos)
                    estructura_CSV["aristas"].append(aristas)
                    estructura_CSV["fuente"].append(fuente)
                    estructura_CSV["sumidero"].append(sumidero)
                    estructura_CSV["densidad"].append(nx.density(grafo_temp))
                    estructura_CSV["media"].append(round(media, 5))
                    estructura_CSV["mediana"].append(round(stats.median(
    tabla_tiempo_ejec), 5))
                    estructura_CSV["varianza"].append(round(stats.pvariance(
    tabla_tiempo_ejec, mu=media), 5))
                    estructura_CSV["desviacion"].append(round(stats.pstdev(
    tabla_tiempo_ejec, mu=media), 5))
```

Tarea4csvfinal.py

Con los datos generados se pasa a realizar el análsis estadístico de los mismos.

3. Análisis de varianza (ANOVA), prueba de *Tukey* y relación general entre factores

Para realizar el análisis del comportamiento de la variable dependiente *tiempo de ejecución* con respecto a cada factor a analizar se realizó un análisis de varianza (ANOVA) para cada factor.

En el caso del análisis de la densidad vs *tiempo de ejecución* se convirtieron los valores de densidad a rangos de valores genéricos, para ello se realizó un histograma para dividir los rangos y llevarlos a una escala cualitativa en correspondencia con el arreglo obtenido del*bins*. El arreglo se muestra en el cuadro siguiente y el histograma en la figura 1 de la página 5

Visualmente del histograma es complejo identificar los rangos correctos para establecer la escala, por lo que se trabajó con el arreglo de rangos que devuelve el histograma siguiente:

Cuadro 1: Arreglo para los rangos de densidad

0.0656	0.0713	0.077	0.0827

A continuación se muestran los cuadros que resumen el resultado del ANOVA para cada factor:

Al analizar los resultados individualmente se observa que en todos los casos el valor de p-valor es menor que el valor de alfa de 0,05. Por lo que en todos los casos se rechaza la hipótesis nula y se acepta la hipótesis alternativa, la que afirma que existen diferencias entre las medias globales de cada factor y las medias de cada grupo por factor para los cutro casos analizados. Lo anterior se refleja en los siguientes diagramas de cajas realizados para cada caso que se reflejan en la figura 2 de la página 7 . De estos se reafirma la idea de que existe diferencia entre las medias de cada grupo por

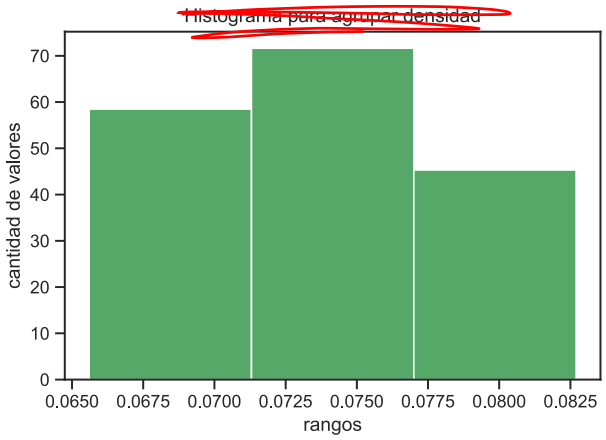


Figura 1: Histograma para determinar escala de densidad

Cuadro 2: ANOVA del tiempo de ejecución vs algoritmo de flujo

Source	SS	DF	MS	F	p-unc	np2
algoritmo_flujo	1.76	2	0.88	3.468	0.031765	0.004
Within	456.998	1797	0.254	-	-	-

Cuadro 3: ANOVA del tiempo de ejecución vs densidad del grafo

Source	SS	DF	MS	F	p-unc	np2
convlogdensidad	61.569	2	30.784	139.629	4.31E-57	0.135
Within	396.49	1797	0.22	-	-	-

Cuadro 4: ANOVA del tiempo de ejecución vs generador

Source	SS	DF	MS	F	p-unc	np2
generador	3.095	2	1.547	6.116	0.002253	0.007
Within	454.664	1797	0.253	-	-	-

Cuadro 5: ANOVA del tiempo de ejecución vs vértices

Source	SS	DF	MS	F	p-unc	np2
vertices	433.225	3	144.408	10571.46	0	0.946
Within	24.534	1796	0.014	-	-	-

factor por lo que sí se puede decir que existe una influencia de cada factor en la variable dependiente *tiempo de ejecución*. Al analizar cada factor se pudiera decir que para el 95 % de las veces como nivel de confianza y un margen de error de 0,05, al realizar el experimento para evaluar cuanto influye el algoritmo de flujo máximo seleccionado en el tiempo promedio de ejecución, el algoritmo que más influye es el *Boykov Kolmogorov*. En el caso del generador de grafos, aparentemente el que más influye en el *tiempo de ejecución* es el Watts Strogatz Newman, para el factor cantidad de vértices, la mayor influencia en el tiempo de ejecución está en el caso que poseen mayor cantidad de vértices, y según la densidad, influye más el rango medio de densidad.

Para tener más certeza sobre cuál de las categorías por factor es la que más influye en la variable dependiente, es recomendable aplicar después del análisis ANOVA una prueba de rango múltiple, en este caso se aplicó la prueba de *TUKEY*, para comprobar la hipótesis por pares en cada grupo de factor. A continuación se muestran los cuadros con los resultados para cada grupo por factor.

Cuadro 6: *Tukey* para factor: Algoritmo de Flujo

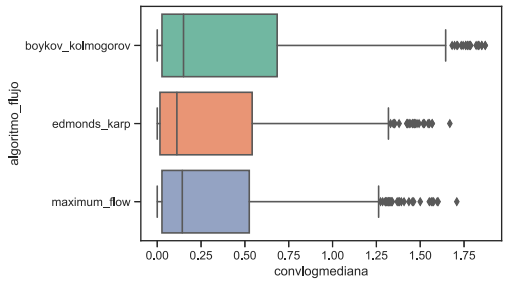
group1	***group2***	***meandiff***	***lower***	***upper***	***reject***
boykov_kolmogorov	edmonds_karp	-0.0621	-0.1304	0.0061	***False***
boykov_kolmogorov	**maximum_flow**	**-0.0699**	-0.1381	-0.0016	***True***
edmonds_karp	maximum_flow	-0.0077	-0.0759	0.0605	***False***

Cuadro 7: *Tukey* para factor: Densidad

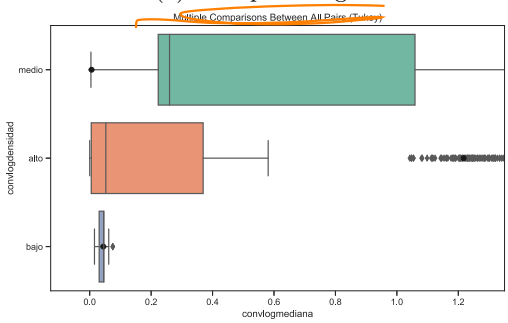
group1	***group2***	***meandiff***	***lower***	***upper***	***reject***
alto	bajo	-0.3101	-0.3851	-0.235	***True***
alto	medio	0.2192	0.1623	0.276	***True***
bajo	**medio**	**0.5292**	0.4538	0.6047	***True***

Cuadro 8: *Tukey* para factor: Generador grafo

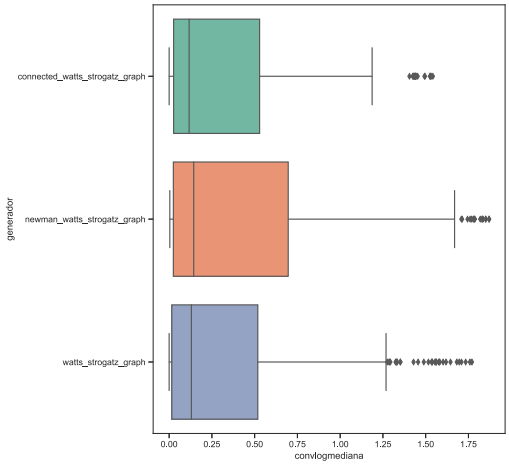
group1	***group2***	***meandiff***	***lower***	***upper***	***reject***
connected_watts_strogatz_graph	**newman_watts_strogatz_graph**	**0.0948**	0.0267	0.163	***True***
connected_watts_strogatz_graph	watts_strogatz_graph	0.0159	-0.0522	0.084	***False***
newman_watts_strogatz_graph	watts_strogatz_graph	-0.0789	-0.147	-0.0108	***True***



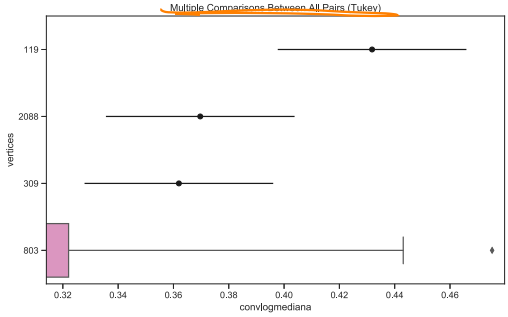
(a) Tiempo vs algoritmo



(b) Tiempo vs densidad del grafo



(c) Tiempo vs generador



(d) Tiempo vs vértices

Figura 2: Diagramas de cajas por factor vs tiempo de ejecución

Cuadro 9: *Tukey* para factor: Cantidad de vértices

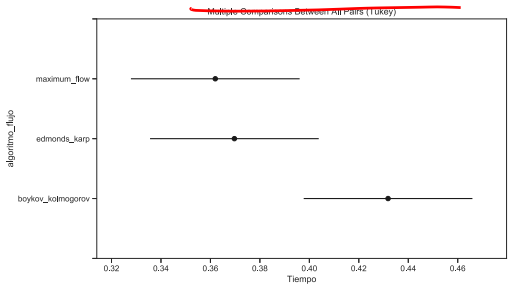
group1	*group2*	*meandiff*	*lower*	*upper*	*reject*
119	**2088**	**1.2113**	1.1913	1.2314	***True***
119	309	0.0384	0.0183	0.0584	***True***
119	803	0.2773	0.2573	0.2974	***True***
2088	309	-1.1729	-1.193	-1.1529	***True***
2088	803	-0.934	-0.954	-0.9139	***True***
309	803	0.239	0.2189	0.259	***True***

De estas tablas se analizan aquellos pares cuyos valores de diferencia de medias sean mayores y se devuelve cual es del par el que más influye. El mismo resultado se puede apreciar mejor en la figura 3 de la página 9.

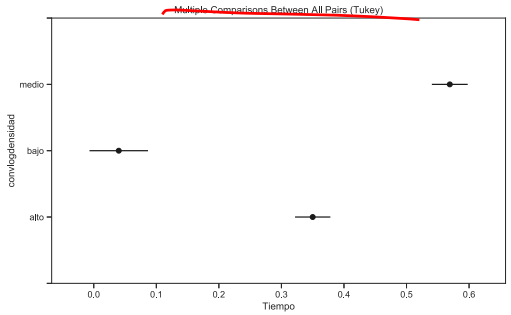
Del análisis del algoritmo de flujo se aprecia que se mantiene el algoritmo de *Boykov Kolmogorov* como el de intervalo de desviación más amplio, y por tanto afecta más, en el par donde se combina. Para la densidad del grafo se aprecia que a pesar de existir diferencia en todos los pares, el que más influye es de la combinación medio con bajo niveles de densidad, se aprecia que el bajo tiene un intervalo mayor, pero el medio posee valores más altos, lo que significa que existe más diferencia entre los valores del rango bajo, pero los de nivel medio influyen más. Para el caso del generador de grafos, se aprecia similitud en los intervalos, por lo que más influye el *Watts Strogatz Newman*. Según la cantidad de vértices en la prueba de *Tukey* no existe diferencia significativa entre los límites superiores e inferiores, y se mantiene que más influye el de mayor cantidad de vértices.

Para conocer la relación entre todas las variables y su influencia en el tiempo de ejecución se realiza la interacción entre las ANOVAS de cada factor, los resultados muestran que los factores que más influyen en el tiempo de ejecución son el generador de grafos y el algoritmo seleccionado, en estos es donde el p valor es menor que 0,05 por tanto se rechaza la hipótesis nula, aceptando la hipótesis alternativa. Los resultados globales se muestran en el cuadro siguiente:

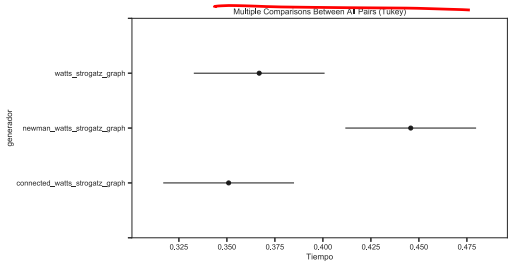
El código que se empleó para realizar el análisis estadístico es el siguiente:



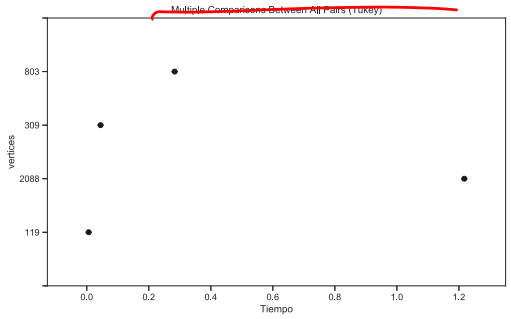
(a) *Tukey*:Tiempo vs algoritmo



(b) *Tukey*:Tiempo vs densidad del grafo



(c) *Tukey*:Tiempo vs generador



(d) *Tukey*:Tiempo vs vértices

Figura 3: Intervalos de confianza de *Tukey*

Cuadro 10: Interacción entre ANOVAS de un factor vs tiempo de ejecución

	sum_sq	df	F	PR(¿F)
generador	0.49851765	2	33.2428814	9.57E-09
algoritmo_flujo	1.76023524	2	117.378775	1.33E-48
vertices	-1.16E-11	3	-5.17E-10	1
convlogdensidad	-1.62E-11	2	-1.08E-09	1
generador:algoritmo_flujo	0.00835511	4	0.27857381	0.8919532
algoritmo_flujo:vertices	2.60338422	6	57.877735	1.44E-65
vertices:convlogdensidad	19.3562824	6	430.247055	5.04E-210
generador:vertices	48.5962079	6	1080.18775	0
generador:convlogdensidad	0.47147159	4	15.7196782	4.32E-10
Residual	13.3091476	1775		

```python
import csv
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
```

Tarea4Estadisticsfinal.py

```python
                                                    'mediana':
    np.float64})
#mejorar los valores de la mediana
logX = np.log1p(df['mediana'])
df = df.assign(convlogmediana=logX.values)
df.drop(['mediana'], axis= 1, inplace= True)

#agrupar los valos de densidad y convertirlo en grupos de baja, media y alta
    densidad


logX = np.log1p(df['densidad'])
df = df.assign(convlogdensidad=logX.values)
df.drop(['densidad'], axis= 1, inplace= True)


his = plt.hist(round(df["convlogdensidad"],4),bins=3, density=True, facecolor='g',
    alpha=0.75)
plt.xlabel ("rangos")
plt.ylabel("cantidad de valores")
plt.title("Histograma para agrupar densidad")
plt.savefig("Imagenes/Histogramadensidad"+".png")
plt.savefig("Imagenes/Histogramadensidad"+".eps")

```

```
print("bins s"
```

Tarea4Estadisticsfinal.py

```
anova_factores=["generador","algoritmo_flujo","vertices","convlogdensidad"]
plt.figure(figsize=(8, 10))
for i in anova_factores:

    print(rp.summary_cont(df['convlogmediana'].groupby(df[i])))

    anovaporfactor = pg.anova (dv='convlogmediana', between=i, data=df, detailed=
    True , )
    pg._export_table (anovaporfactor,("estadisttable/Tabla_ANOVA"+i+".csv"))

    ejes=sns.boxplot(x=df["convlogmediana"], y=df[i], data=df, palette="Set2")
    plt.savefig("Imagenes/aboxplot"+ i+".png", bbox_inches='tight')
    plt.savefig("Imagenes/aboxplot" + i + ".eps", bbox_inches='tight')
    tukey = pairwise_tukeyhsd(endog = df["convlogmediana"],      # Data
                          groups= df[i],   # Groups
                          alpha=0.05)          # Significance level

    tukey.plot_simultaneous(xlabel='Tiempo', ylabel=i)    # Plot group confidence
    intervals
#    plt.vlines(x=49.57,ymin=-0.5,ymax=4.5, color="red")

    plt.savefig("Imagenes/tablatukey"+ i+".png", bbox_inches='tight')
    plt.savefig("Imagenes/tablatukey" + i + ".eps", bbox_inches='tight')
    print(tukey.summary())

    excel_tukey = open("estadisttable/Tukey"+i+".csv", 'w')
    with excel_tukey:
        writer = csv.writer(excel_tukey)
        writer.writerows(tukey.summary())

model_name = ols('convlogmediana ~ generador+algoritmo_flujo+vertices+
    convlogdensidad +generador*algoritmo_flujo+algoritmo_flujo*vertices+vertices*
    convlogdensidad+generador*vertices+generador*convlogdensidad+algoritmo_flujo+
    vertices*convlogdensidad', data=df).fit()
model_name.summary()
aov_table = sm.stats.anova_lm(model_name, typ=2)
df1=pd.DataFrame(aov_table)
df1.to_csv("multianova.csv")

plt.show()
print ("fin")
```

Tarea4Estadisticsfinal.py

Referencias

[1] Leonardo J. Caballero. *Materiales del entrenamiento de programación en Python*. 2019.

[2] Steven H. Strogatz Duncan J. Watts. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, jun. 1998. doi: 10.1038/30918. URL https://worrydream.com/refs/Watts-CollectiveDynamics-OfSmallWorldNetworks.pdf.

and

[3] Desarrolladores NetworkX. https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.flow.boykov_kolmogorov.html, . Accessed:2019-03-31.

[4] Desarrolladores NetworkX. ttps://https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.flow.edmonds_karp.html, . Accessed:2019-03-31.

[5] Desarrolladores NetworkX. ttps://https://networkx.github.io/documentation/stable/reference/generators.html, . Accessed: 2019-03-29.

[6] Desarrolladores NetworkX. ttps://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.flow.maximum_flow.html, . Accessed: 2019-03-17.

Notes

10-2