Tarea 4

5271

2 de abril de 2019

1. Algoritmos generadores de grafos.

Gracias a la versatilidad de Los grafos como modelo de representación de datos, los procesos aleatorios de generación de grafos también son relevantes en aplicaciones que van desde la física, biología a la sociología[8] .

Para realizar la tarea se seleccionaron tres algoritmos generadores de grafos de la biblioteca de networkx, con el objetivo de crear los grafos con pesos con distribución normal (como se muestra en la figura 1de la página 2) a los que se le aplicaran los algoritmos de flujo máximo para realizar los experimentos requeridos. Los tres algoritmos escogidos son de generación aleatorias.

Los algoritmos escogidos fueron los siguientes:

- *Erdos renyi graph* (en el modelo *G (n, p)*, el grafo se construye conectando los n vértices al azar. Cada arista se incluye en el grafo con probabilidad p independiente de cualquier otro borde) [3].
- *Fast gnp random graph* (este algoritmo recibe como parámetros n números de vértices y probabilidad p de ocurrencia de aristas, el mismo devuelve un grafo aleatorio) [4].
- *Binomial graph* (este algoritmo recibe como parámetros n números de vértices y probabilidad p de ocurrencia de aristas, el mismo devuelve un grafo aleatorio, para grafos dispersos (para valores pequeños de p), *Fast gnp random graph* es un algoritmo más rápido) [5].

  En la figura 2 de la 3 se muestra ejemplos de grafos generados por los algoritmos antes mencionados.

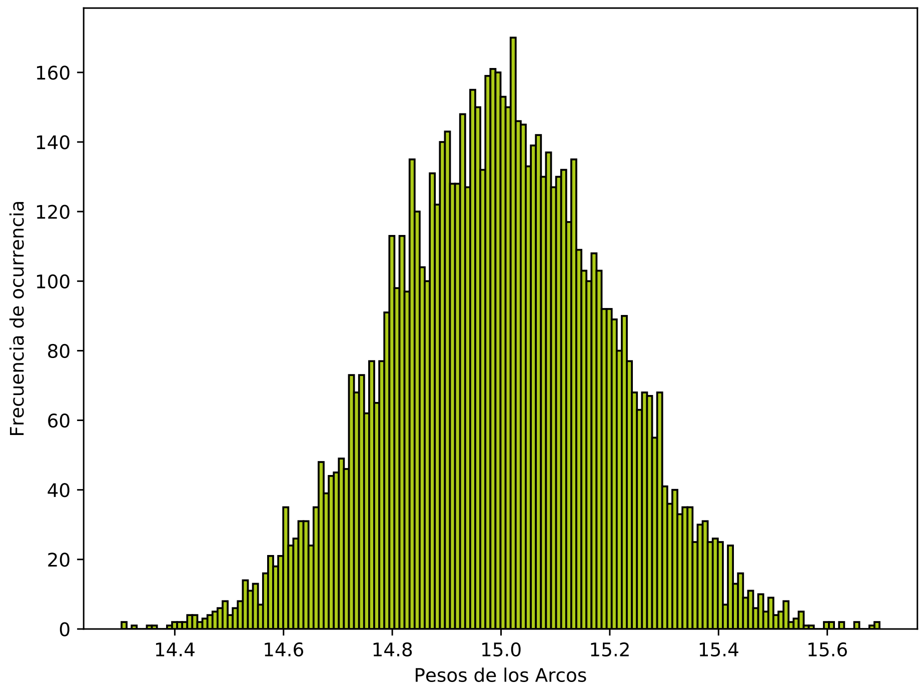


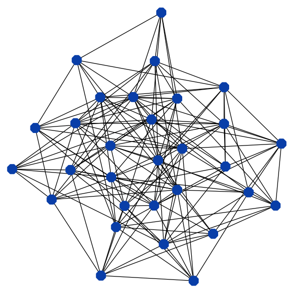
Figura 1: Histograma de distribución de los pesos.

2. Algoritmos de flujo máximo.

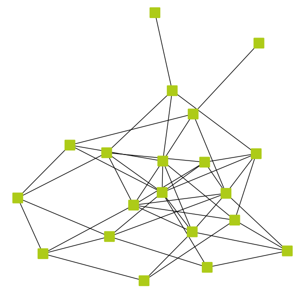
En los problemas de flujo en redes, las aristas representan vías por las que puede circular elementos: datos, agua, corriente eléctrica, entre otras. Los pesos de las aristas representan la capacidad máxima de una vía: velocidad de una conexión, volumen máximo de agua, voltaje de una línea eléctrica, entre otras; aunque es posible que la cantidad real de flujo sea menor.

El problema del flujo máximo consiste en lo siguiente: dado un grafo con pesos, $G = (V, A, W)$, que representa las capacidades máximas de los canales, un vértice fuente f y otro sumidero s en V, encontrar la cantidad máxima de flujo que puede circular desde f hasta s.

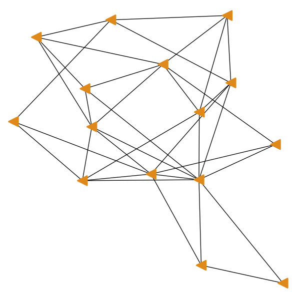
En la librería de networkx encontramos varios algoritmos con los que podemos atacar los problemas de flujo máximo, para la realización de esta tarea se escogerán tres algoritmos de dicha librería.



(a) *Binomial graph*, con 30 vértices



(b) *Erdos renyi graph*, con 20 vértices



(c) *Fast gnp random graph*, con 15 vértices

Figura 2: Ejemplo de grafos generados con los algoritmos seleccionados.

Los algoritmos escogidos fueron los siguientes:

- *Shortest augmenting path* (es uno de los enfoques más clásicos para la máxima coincidencia y los problemas de flujo máximo. Sorprendentemente, aunque esta idea es una de las técnicas más básicas, está lejos de ser completamente entendida. Es más fácil hablar de ello introduciendo el problema de emparejamiento bipartito en línea[1]. Este algoritmo encuentra el flujo máximo de un solo producto utilizando la ruta de aumento más corto y devuelve la red residual resultante después de calcular el flujo máximo).
- *maximum flow* (Encuentra la ruta por la cual pasa la máxima cantidad de flujo, recibe como parámetros un grafo G, una fuente f, un sumidero s y además una capacidad que de no tenerla, se considera que el borde tiene una capacidad infinita. Se puede aplicar en grafos tanto dirigidos como no dirigidos.) [6].

- *Preflow push* (encuentra un flujo máximo de un solo producto utilizando el algoritmo de empuje previo al flujo de la etiqueta más alta. Esta función devuelve la red residual resultante después de calcular el flujo máximo. Este algoritmo tiene un tiempo de ejecución de $O(n^2\sqrt{m})$ para n vértices y m aristas.) [7].

3. Generación de datos.

Con el objetivo de realizar las mediciones de los tiempos de ejecución de los algoritmos de flujo máximo seleccionados se desarrolló el siguiente código.

En primer lugar, se crea una función ($Algoritmo_F M()$) que recibe como parámetros el algoritmo de flujo máximo ($algoritmoF$) que se va a utilizar, el grafo al que se le aplicara el algoritmo($Graf$), la fuente ($fuente$) y sumidero ($sumidero$). Al llamar esta función se genera con cada ejecución los datos que son guardados en un *data frame* del cual se muestra un fragmentó en el cuadro 1 de la página 5.

```
def Algoritmo_FM(algoritmoF,Graf, funte, sumidero,):
    tiempos_ejecucion = []
    for medicion in range(1, mediciones + 1):
        t_inicio = dt.datetime.now()
        obj = algoritmos_flujo[algoritmoF](Graf, funte, sumidero, capacity="capacity")
        t_fin = dt.datetime.now()
        tiempo_consumido_segundos = (t_fin - t_inicio).total_seconds()
        tiempos_ejecucion.append(tiempo_consumido_segundos)
    media = stats.mean(tiempos_ejecucion)

    t_csv["grafo"].append("vertices" + str(inst_g_n) + "aristas" + str(aristas))
    t_csv["generador_grafo"].append(generador_grafo)
    t_csv["vertices"].append(inst_g_n)
    t_csv["densidad"].append(nx.density(Grafo))
    t_csv["aristas"].append(aristas)
    t_csv["f"].append(f)
    t_csv["s"].append(s)
    t_csv["algoritmo_fm"].append(algoritmo)
    t_csv["media"].append(round(media, 5))
    t_csv["mediana"].append(round(stats.median(tiempos_ejecucion), 5))
    t_csv["varianza"].append(round(stats.pvariance(tiempos_ejecucion, mu=media), 5))
    t_csv["desv"].append(round(stats.pstdev(tiempos_ejecucion, mu=media), 5))
    return t_csv
```

Generar_datos.py

Cuadro 1: Fracmento de *data frame* generado.

grafo	generador	algoritmo_fm	vertices	densidad	aristas	f	s	mediana	varianza	desv
v256a8140	fast_gnp	shortest_a_path	256	0.24939	8140	33	101	0.06247	0.00004	0.00618
v256a8141	fast_gnp	edmonds_karp	256	0.24939	8140	33	101	0.04692	0.00001	0.00318
v256a8142	fast_gnp	preflow_push	256	0.24939	8140	33	101	0.08225	0.00025	0.01596
v256a8143	fast_gnp	shortest_a_path	256	0.24939	8140	33	101	0.06251	0.00004	0.00638
v256a8144	fast_gnp	edmonds_karp	256	0.24939	8140	33	101	0.05291	0.00007	0.00834
v256a8145	fast_gnp	preflow_push	256	0.24939	8140	33	101	0.08253	0.00017	0.01296
v256a8146	fast_gnp	shortest_a_path	256	0.24939	8140	33	101	0.06903	0.00020	0.01398
v256a8147	fast_gnp	edmonds_karp	256	0.24939	8140	33	101	0.05296	0.00003	0.00566

En este otro fragmentó del código es donde se generan los grafos y se llama la función ($Algoritmo_F M()$).

```
numero_intancias = 10
mediciones = 5
archivo_CSV = "Datost4.csv"
control_iteraciones = 0

generadores_grafos = {
"fast_gnp_random_graph": nx.fast_gnp_random_graph,
"binomial_graph": nx.binomial_graph,
"erdos_renyi_graph": nx.erdos_renyi_graph
}

algoritmos_flujo = {
"shortest_augmenting_path": shortest_augmenting_path,
"maximum_flow": maximum_flow,
"preflow_push": preflow_push
}

t_csv = {
"grafo": [],"generador_grafo": [],
"algoritmo_fm": [],"vertices": [],
"densidad": [],"aristas": [],
"f": [],"s": [],"media": [],
"mediana": [],"varianza": [],
"desv": []
}
for generador_grafo in generadores_grafos:
    for inst_g_n in [round(pow(2, value + 1))
                     for value in
                     range(7, 11 )]:


        for grafo in range(1, numero_intancias + 1):
            f = np.random.randint(1, high=(inst_g_n - 1), dtype="
    int")
            s = np.random.randint(1, high=(inst_g_n - 1), dtype="
    int")

            while s == f:
```

```
                f = np.random.randint(1, high=(inst_g_n - 1), dtype="int")
                s = np.random.randint(1, high=(inst_g_n - 1), dtype="int")
            Grafo = generadores_grafos[generador_grafo](inst_g_n, 0.25, seed=None)
            aristas = Grafo.number_of_edges()
            p_n_distribuidos = np.random.normal(15, 0.2, aristas)
            incremento = 0

        for (u, v) in Grafo.edges():
            Grafo.edges[u, v]["capacity"] = p_n_distribuidos[incremento]
            incremento += 1
        pesos = []
        for (u, v) in Grafo.edges():
            t = Grafo.edges[u, v]["capacity"]
            pesos.append(t)
        plt.figure(figsize=(8, 6))
        n = plt.hist(pesos, bins =70, color="#932525", alpha=1, edgecolor = 'black', linewidth=1)
        plt.xlabel('Pesos de los Arcos')
        plt.ylabel('Frecuencia de ocurrencia')

        plt.savefig("histograf.png")
        plt.savefig("histograf.eps")
        for instancia_grafo in range(1, 6):
            for algoritmo in algoritmos_flujo:
                d= Algoritmo_FM(algoritmo, Grafo, f, s, control_iteraciones)
                control_iteraciones=d
ds = pd.DataFrame(t_csv)
ds.to_csv(archivo_CSV, encoding="utf-8", index=None)
```

Generar_datos.py

4. Resultados del Análisis de los datos.

Con los datos recopilado de las ejecuciones de los tres algoritmos de flujo máximo con los ciento veinte grafos y sus cinco combinaciones de fuentes y sumidero, se realizó una serie de diagramas y pruebas estadísticas. A continuación, se muestra un el fragmento de código donde se lleva a cabo lo antes mencionado.

```
df = pd.read_csv("Datost4.csv", index_col=None, usecols=[1,2,3,4,9], dtype={'generador_grafo': 'category',

        'algoritmo_fm': 'category','vertices': 'category','densidad':np.float64, 'mediana': np.float64} )
modelo = ols('mediana_log ~ generador_grafo+algoritmo_fm+vertices+densidad+generador_grafo*algoritmo_fm+algoritmo_fm*vertices+vertices*densidad+generador_grafo*vertices+generador_grafo*densidad+algoritmo_fm*densidad',data=df).fit()
```

```python
print(modelo.summary())
modelo_csv = open("Anova_Mult.csv", 'w')
aov_table = sm.stats.anova_lm(modelo, typ=2)
df1=pd.DataFrame(aov_table)
df1.to_csv("modelo.csv")
for column in range(0, df["densidad"].count()):
    pass
    if  df.iat[column, 3] >=0.2061  and df.iat[column, 3] < 0.20854:
        df.iat[column, 3] = 1
    elif df.iat[column, 3] >=0.20854 and df.iat[column, 3] < 0.21098:
        df.iat[column, 3] = 2
    else:
        df.iat[column, 3] = 3
print(df["densidad"])
df['densidad'].replace({1:"baja", 2: 'media', 3:'alta' }, inplace=True)
print(df["densidad"])
logX = np.log1p(df['mediana'])
df = df.assign(mediana_log=logX.values)
df.drop(['mediana'], axis= 1, inplace= True)

factores=["vertices","generador_grafo","densidad","algoritmo_fm"]
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,("ANOVAs"+i+".csv"))

    ax=sns.boxplot(x=df["mediana_log"], y=df[i], data=df, palette="cubehelix")

    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='Tiempo', ylabel=i)
    plt.vlines(x=49.57,ymin=-0.5,ymax=4.5, color="red")
    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()
```

Analisis_datos.py

4.1. Análisis de varianza(ANOVA).

El análisis de varianza (ANOVA) es la técnica central en el análisis de datos experimentales. La idea general de esta técnica es separar la variación total en las partes con las que contribuye cada fuente de variación en el experimento. En el caso de los diseños completamente al azar se separan la variabilidad debida a los tratamientos y la debida al error. Cuando la primera predomina sobre la segunda, es cuando se concluye que las medias son diferentes. Cuando los tratamientos no dominan contribuyen igual o menos que el error, por lo que se concluye que las medias son iguales [2].

Para analizar si los diferentes factores (algoritmo generador de grafo, algoritmo de flujo máximo, numero de vértices y densidad del grafo)influían en la variable dependiente *tiempo de ejecución* se realizó un ANOVA de un factor para cada caso.

4.1.1. Influencia del algoritmo generador de grafos en el tiempo de ejecución.

El siguiente cuadro muestra el resultado de la aplicación del ANOVA.

Cuadro 2: ANOVA, relación del algoritmo generador con el *tiempo de ejecución*

Factor	**SS**	**DF**	**MS**	**F**	**p-unc**	**np2**
generador_grafo	0.280	2	0.140	0.354	0.702	0
Within	712.085	1797	0.396	-	-	-

En el cuadro 2 se muestra que no existen diferencia entre las medianas de los grupos de factores ya que el $p - unc$ es mayor que 0,05 por lo que se acepta la hipótesis de que el tipo de generador no influye en el *tiempo de ejecución*. Esto se puede observar en la figura 4 de la página 10.

4.1.2. Influencia del algoritmo de flujo máximo en el tiempo de ejecución.

El siguiente cuadro muestra el resultado de la aplicación del ANOVA.

En el cuadro 3 se muestra que no existen diferencia entre las medianas de los grupos de factores ya que el $p - unc$ es mayor que 0,05 por lo que se acepta la

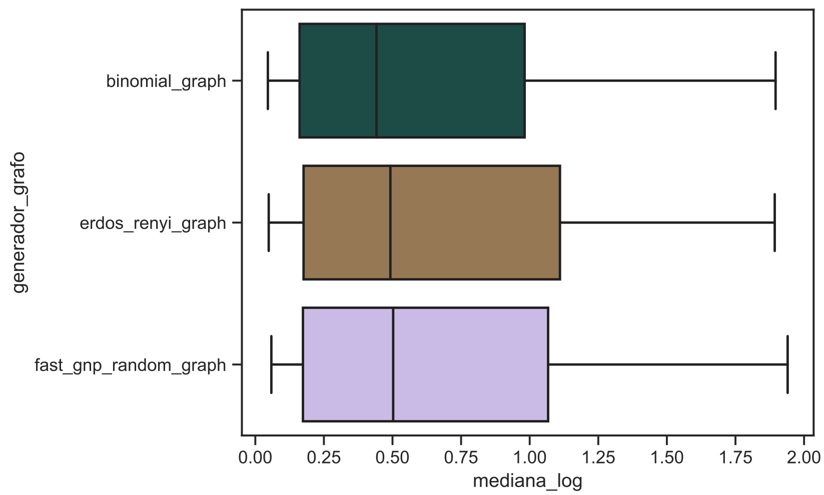


Figura 3: Diagrama de caja y bigotes que relaciona los tiempos de ejecución con los algoritmos generadores de grafos.

Cuadro 3: ANOVA, relación del algoritmo de flujo máximo con el *tiempo de ejecución*

Factor	**SS**	**DF**	**MS**	**F**	**p-unc**	**np2**
algoritmo_fm	1.499	2	0.749	1.895	0.151	0.002
Within	710.867	1797	0.396	-	-	-

hipótesis de que el tipo de algoritmo no influye en el *tiempo de ejecución*. Esto se puede observar en la figura **??** de la página **??**.

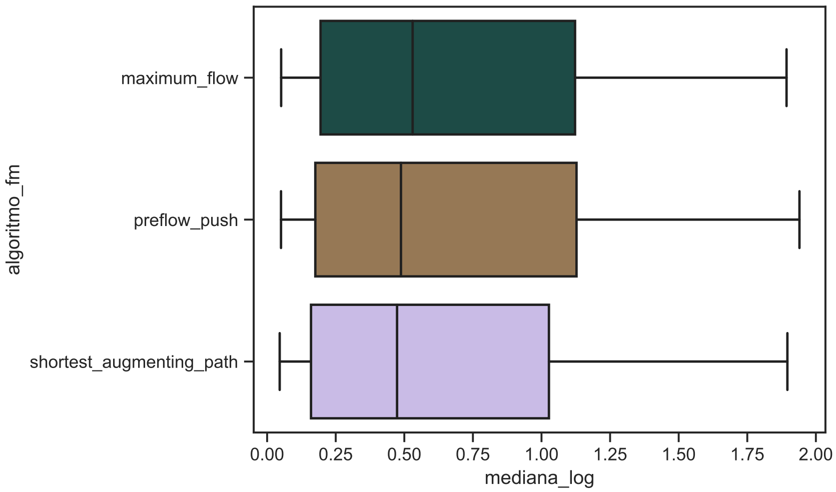


Figura 4: Diagrama de caja y bigotes que relaciona los tiempos de ejecución con los algoritmos de flujo máximo.

4.1.3. Influencia del número de vértices en el tiempo de ejecución.

El siguiente cuadro muestra el resultado de la aplicación del ANOVA.

Cuadro 4: ANOVA, relación del número de vértices con el *tiempo de ejecución*

Factor	**SS**	**DF**	**MS**	**F**	**p-unc**	**np2**
vértices	706.976	3	235.659	78536.187	0	0.992
Within	5.389	1796	0.003	-	-	-

En el cuadro 4 se muestra que existen grandes diferencia entre las medianas de los grupos de factores ya que el $p - unc$ es menor que 0,05 por lo que se se rechaza la hipótesis de que la cantidad de vértices no influye en el *tiempo de ejecución*. Esto se puede observar en la figura 5 de la página 11. Por tal motivo se realiza la prueva de Tukey mostrara las diferencias entre las medianas de los factores, lo que se evidencia en el cuadro 5 de la página 11.

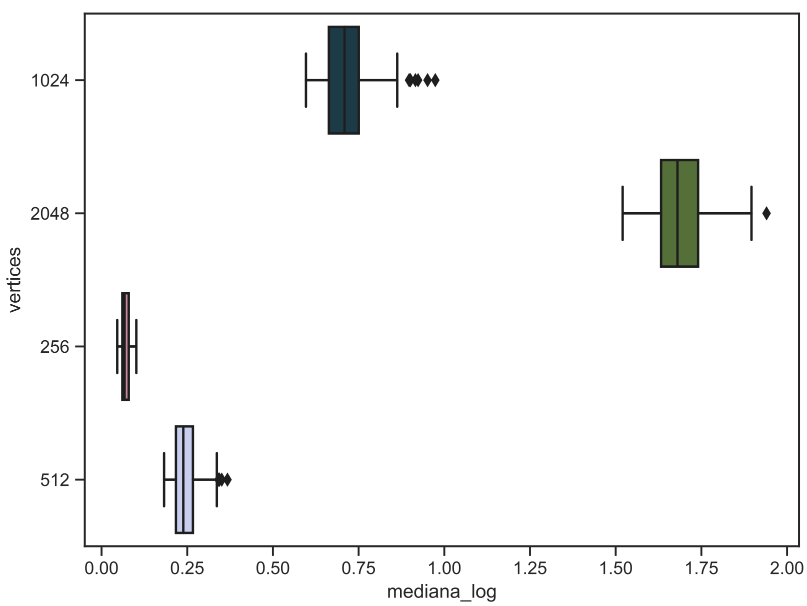


Figura 5: Diagrama de caja y bigotes que relaciona los tiempos de ejecución con el número de vértices.

En el cuadro 5 se puede observar que en todos los casos se rechaza la hipótesis.En la figura 6 de la página 12 muestra claramente este hecho.

Cuadro 5: Tukey, influencia del número de vértices en el tiempo de ejecución.Add caption

group1	***group2***	***meandiff***	***lower***	***upper***	***reject***
1024	2048	0.97	0.9606	0.9794	*True*
1024	256	-0.6445	-0.6539	-0.6352	*True*
1024	512	-0.4689	-0.4782	-0.4595	*True*
2048	256	-1.6146	-1.624	-1.6052	*True*
2048	512	-1.4389	-1.4483	-1.4295	*True*
256	512	0.1757	0.1663	0.1851	*True*

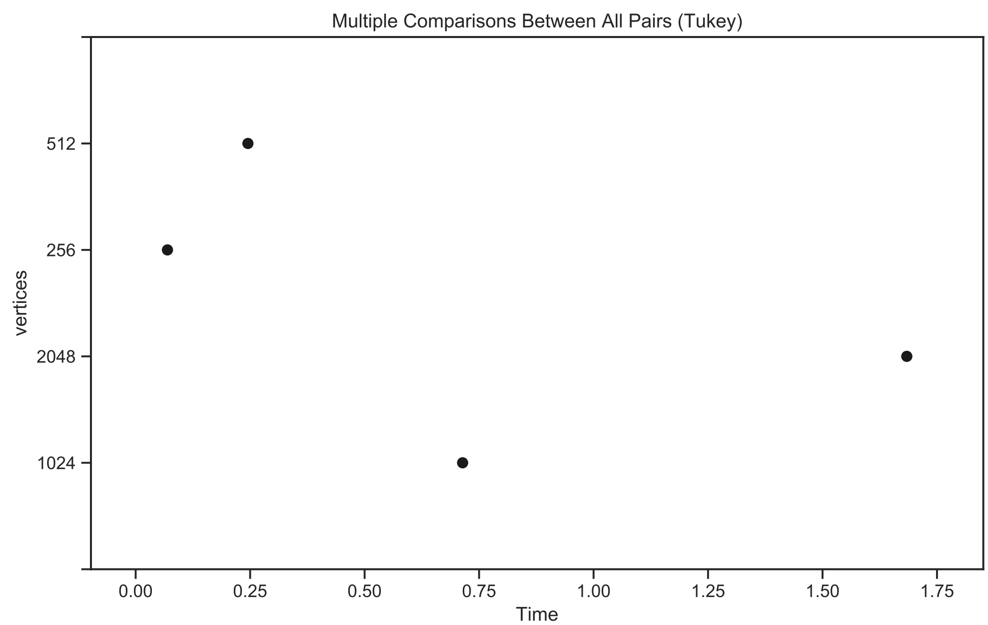


Figura 6: Diagrama simultaneo que relaciona los tiempos de ejecución con los grupos del factor número de vértices.

4.1.4. Influencia de la densidad de los grafos en el tiempo de ejecución.

En el caso del factor densidad se hizo una categorización por rangos para hacer cómoda la visualización de su relación con la variable dependiente estos rangos se obtuvieron a través de los contenedores que devolvió el histograma que se muestra en la figura 7 de la página 13.

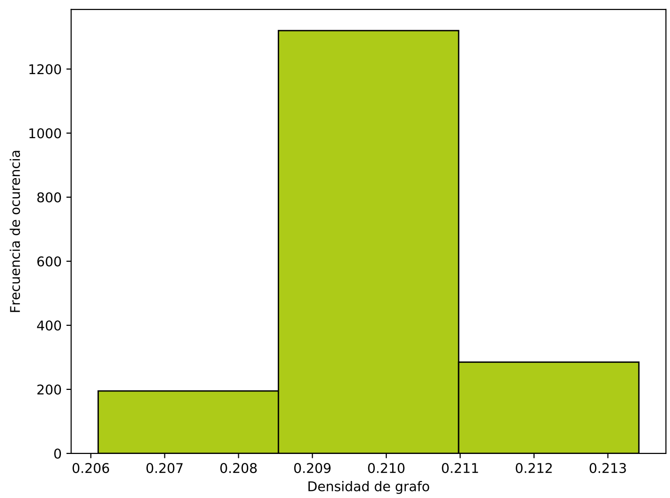


Figura 7: Histograma de densidad de los grafos.

El siguiente cuadro muestra el resultado de la aplicación del ANOVA. En el

Cuadro 6: ANOVA, influencia de la densidad de los grafos en el tiempo de ejecución.

Factor	**SS**	**DF**	**MS**	**F**	**p-unc**	**np2**
densidad	170.777	2	85.388	283.320	0.000	0.24
Within	541.589	1797	0.301	-	-	-

cuadro 7 se muestra que como en elcuadro 4 existen grandes diferencia entre las medianas de los grupos de factores ya que el $p - unc$ es menor que 0,05 por lo que se se rechaza la hipótesis de que la densidad de los grafos no influye en el *tiempo de ejecución*. Esto se puede observar en la figura 8 de la página 14. Por tal motivo se realiza la prueva de Tukey mostrara las diferencias entre las medianas de los factores, lo que se evidencia en el cuadro 7 de la página 14.

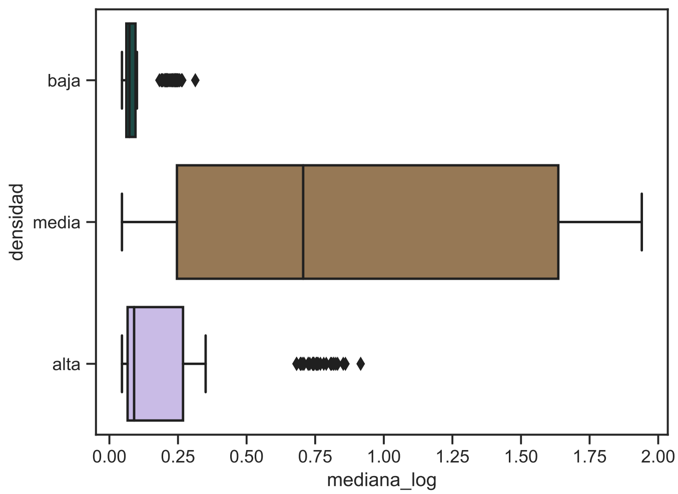


Figura 8: Diagrama de caja y bigotes que relaciona los tiempos de ejecución con la densidad de los grafos.

Cuadro 7: Tukey,influencia de la densidad de los grafos en el tiempo de ejecución.

group1	***group2***	***meandiff***	***lower***	***upper***	***reject***
alta	baja	-0.1085	-0.2281	0.0112	*False*
alta	media	0.6497	0.5656	0.7338	*True*
baja	media	0.7582	0.6594	0.8569	*True*

En el cuadro 7 se puede observar que en dos de los tres casos se rechaza la hipótesis y en el pareo de densidad alta y baja no existen diferencias stadísticas en cuanto a la mediana. En la figura 9 de la página 15 muestra claramente este hecho.

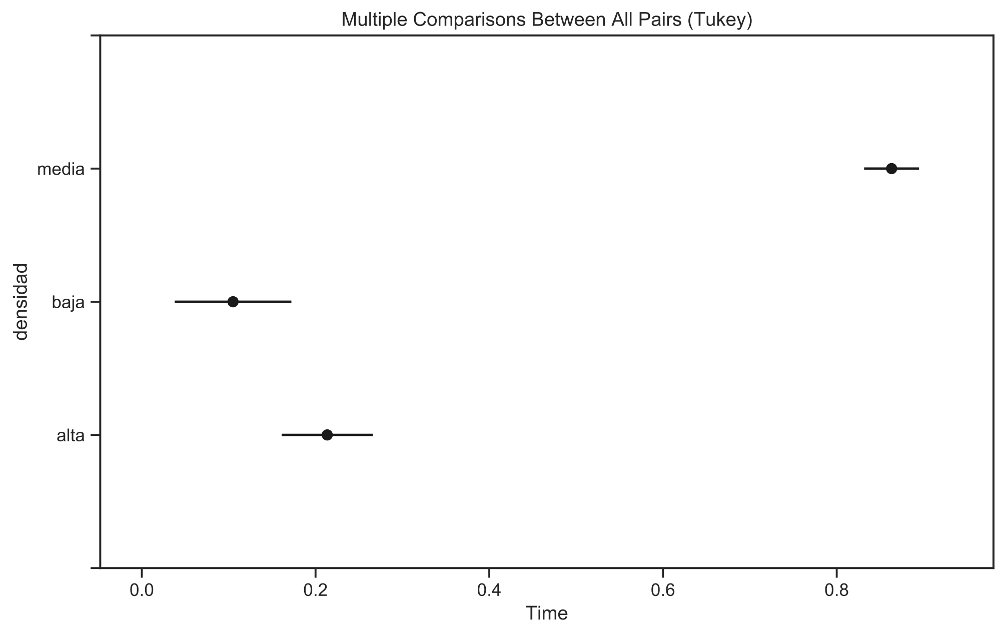


Figura 9: Diagrama simultaneo que relaciona los tiempos de ejecución con los grupos del factor densidad de los grafos.

4.1.5. Influencia de los cuatro factores(algoritmo generador de grafo, algoritmo de flujo máximo, numero de vértices y densidad del grafo) en el tiempo de ejecución.

Para analizar este caso se realizo ANOVA multifactorial dando como resultado el cuadro 8 de la página 16.

Cuadro 8: ANOVA multifactor, influencia de los cuatro factores en el tiempo de ejecución.

	sum_sq	**df**	**F**	**PR(¿F)**
generador_grafo	0.0291	2	33.0499	0.0000
algoritmo_fm	-0.0001	2	-0.1439	1.0000
vertices	46.4767	3	35163.6974	0.0000
densidad	0.0000	2	0.0000	0.9999
generador_grafo:algoritmo_fm	0.0008	4	0.4714	0.7567
algoritmo_fm:vertices	0.0001	6	0.0248	0.8748
vertices:densidad	0.0001	6	0.0425	0.9584
generador_grafo:vertices	0.0651	6	24.6313	0.0000
generador_grafo:densidad	0.0000	4	0.0181	0.8930
algoritmo_fm:densidad	0.0001	4	0.0425	0.9584
Residual	0.7798	1770		

En el cuadro 8 se puede observar que los factores que más influyen en el tiempo de ejecución son el número de vértices y la densidad de los grafos, que además existe una relación entre el número de nodos y la densidad.

Referencias

[1] Bartłomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych-Pawlewicz. Shortest augmenting paths for online matchings on trees. *Theory of Computing Systems*, 62(2):337–348, Feb 2018.

[2] Humberto Gutiérrez and Román de la Vara. *Análisis y diseño de experimentos*. The McGraw-Hill Companies, Inc., segunda edición edition, 2008. 60–74.

[3] Desarrolladores NetworkX. `https://networkx.github.io/documentation/stable/reference/generated/networkx.generators.random_graphs.erdos_renyi_graph.html#networkx.generators.random_graphs.erdos_renyi_graph`. Accessed: 01-04-2019.

[4] Desarrolladores NetworkX. `https://networkx.github.io/documentation/stable/reference/generated/networkx.generators.random_graphs.fast_gnp_random_graph.html#networkx.generators.random_graphs.fast_gnp_random_graph`. Accessed: 01-04-2019.

[5] Desarrolladores NetworkX. `https://networkx.github.io/documentation/stable/reference/generated/networkx.generators.random_graphs.binomial_graph.html#networkx.generators.random_graphs.binomial_graph`. Accessed: 01-04-2019.

[6] Desarrolladores NetworkX. `https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.flow.maximum_flow.html#networkx.algorithms.flow.maximum_flow`. Accessed: 01-04-2019.

[7] Desarrolladores NetworkX. `https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.coloring.greedy_color.html`. Accessed: 18-03-2019.

[8] Sadegh Nobari, Xuesong Lu, Panagiotis Karras, and Stéphane Bressan. Fast random graph generation. In *Proceedings of the 14th International Conference on Extending Database Technology*, EDBT/ICDT '11, pages 331–342, New York, NY, USA, 2011. ACM.