Tarea No.5: Flujo en Redes

Dayli Machado (5275)

30 de abril de 2019

1. Objetivo

Determinar mediante un diseño de experimento, empleando el análisis de varianza de un factor y otras pruebas estadísticas, la influencia que pueden tener en las variables dependientes *tiempo de ejecución del algoritmo de flujo máximo seleccionado y el máximo flujo posible a obtener para diferentes combinaciones de fuentes y sumideros*, en función de las siguientes características estructurales de cada vértice: ditribución de grado, coeficiente de agrupamiento, centralidad de cercanía, centralidad de carga, excentricidad y page rank.

2. Generador de grafos y algoritmo de flujo máximo seleccionado

Para la evaluación anterior se seleccionaron los generadores de grafos aleatorios los cuales según [6] son útiles para comprender los procesos estocásticos que ocurren en una red. Adicionalmente plantea que la generación de grafos aleatorios que controlan algunas condiciones, como la distribución de grado, el coeficiente de agrupamiento y la secuencia de grados, permite estudiar cómo se forman las estructuras de redes de la vida real y plantea como ejemplos prácticos de aplicación, la participación electoral, la propagación de epidemias, así como el comportamiento en redes sociales.

En la evaluación anterior se seleccionaron dentro del grupo de generadores de grafos aleatorios, tres modelos de generadores desarrollados por los autores *Watts–Strogatz*[8]. Estos permiten de una forma relativamente sencilla y con menor número de parámetros generar los grafos. Una propiedad importante de estos tres generadores es que se desarrollan bajo la teoría de red de mundos pequeños. Esta propiedad revela según sus creadores y también a criterio de[5] que la sociedad humana es una red social con forma de mundo pequeño y que están interconectados entre sí, con estructura de red, cuyos nódulos son personas y los enlaces, las interrelaciones entre ellos. 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, reflejando mejor

el comportamiento de fenómenos reales. 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[4].

Para esta tarea de los generadores de grafos empleados en la anterior que fueron los siguientes:

- *grafo de Watts-Strogatz Newman*
- *grafo de Watts-Strogatz*
- *grafo de Watts-Strogatz Conectado*

Se seleccionó de los que menos influían en el tiempo de ejecución el generador de grafo de *Watts-Strogatz Conectado* en combinación con el algoritmo de fujo máximo *Edmons-Karp* de los siguientes algoritmos de flujo máximo empleados:

- *Algoritmo Boykov-Kolmogorov*
- *Algoritmo de Flujo Máximo*
- *Algoritmo Edmonds-Karp*

El algoritmo de *Edmonds-Karp* calcula el flujo máximo de un producto, debe emplearse con grafos dirigidos aunque también se emplea para no dirigidos, requiere asignársele una capacidad sino la toma como infinita [7] y, además, devuelve la red residual de flujo máximo (caraterística que el algoritmo *Flujo Máximo* no posee, pues este lo que devuelve es un arreglo del flujo máximo que pasa por las aristas seleccionadas) de ahí que se seleccionara el *Edmons-Karp* como algoritmo para calcular el flujo máximo en esta evaluación.

El ejemplo de aplicación que combina generador de grafo aleatorio con algoritmo de flujo máximo seleccionados puede ser la propagación de epidemias endémicas entre regiones determinadas, donde cada vértice serían las regiones objeto de estudios y las aristas el número de epidemias endémicas de cada región que se pueden propagar de un lugar a otro.

A continuación se muestra el fragmento de código desarrollado para generar los grafos con la capacidad asignada apoyándose en [1],[3],[2]:

```python
def Generador_grafo(n,k,p):
    #S=nx.watts_strogatz_graph(n, k, p,)
    S=nx.connected_watts_strogatz_graph(n,k,p,tries=100, seed=None)
    scale = 2
    rang = 11
    size = S.number_of_edges()
    e=S.edges(nbunch=None, data=True, default=None)
    X = truncnorm(a=-rang/scale, b=+rang/scale, scale=scale).rvs(size=size)
    X = X.round().astype(int)+rang
    G=nx.Graph()
    count=0;
    for i in e:
        G.add_edge(i[0],i[1],capacity=X[count])
        count+=1
    df = pd.DataFrame()
    df = nx.to_pandas_adjacency(G, dtype=int, weight='capacity')
    df.to_csv("grafo5.csv", index=None, header=None)
    PrintGraph(S,X)

Generador_grafo(20,7,0.5)
```

Tarea5GeneradorGrafos.py

2.1. Generación de datos para el análisis estadístico

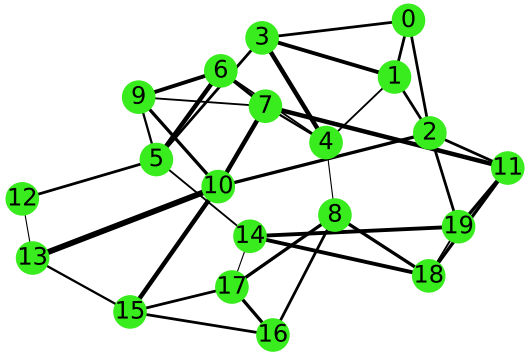
Los grafos se generaron con un código y luego otro los lee y dibuja con el algoritmo de acomodo usado en la tarea anterior y que mejor se ajustaba para esta tarea el *kamada_kawai*. El código usado para ello se muestra a continuación:

```
def lee_grafo(adress):
    ds = pd.read_csv(adress, header=None)
    G = nx.from_pandas_adjacency(ds)
    return G

def calcula_tiempo(G,a,b):
    start_time=time()
#    fm=nx.maximum_flow(G, a, b,capacity="weight")
    for i in range (100):
        fm=edmonds_karp(G, a, b,capacity="weight")
        print(i)
    time_elapsed = time() - start_time

    return time_elapsed

def DibujaGrafo(G):

    pesos=[]
    for edge in G.edges():
        pesos.append(G.edges[edge]['weight'])
    pesos[:] = [x/7*x/10 for x in pesos]

#    pos=nx.spring_layout(G)
    pos=nx.kamada_kawai_layout(G,scale=10)

    nx.draw_networkx_nodes(G, pos, node_size=400, node_color='#38ec1d', node_shape='
    o')
    nx.draw_networkx_edges(G, pos, width=pesos,edge_color='black')
    labels = {}
    for i in G.nodes:
        labels[i]=str(i)
    nx.draw_networkx_labels(G, pos, labels, font_size=15 )

    plt.axis('off')
    plt.savefig("fig5a.png",dpi=600)
    plt.savefig("fig5a.eps",dpi=600)
    df=pd.DataFrame(pos)
    df.to_csv("pos_grafo5.csv", index=None, header=None)
    return pos
```

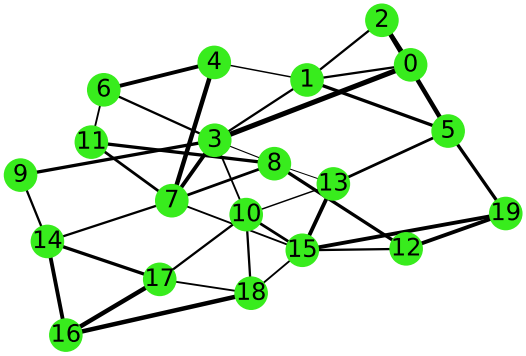
T5ProTimeFLujo.py

Los grafos generados se muestran en la figura 1 de la página 5

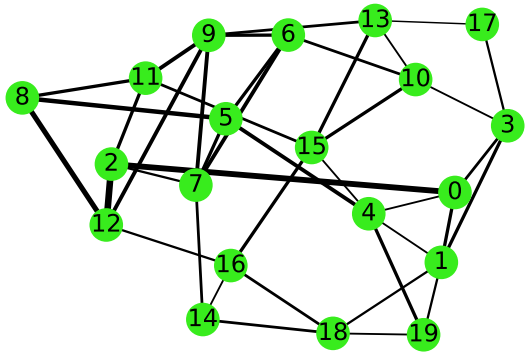
Luego se calcularon las propiedades para cada vértice de cada grafo usando el siguiente código:



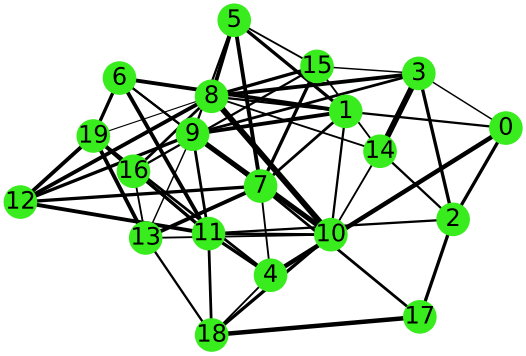
(a) *Grafo 1*



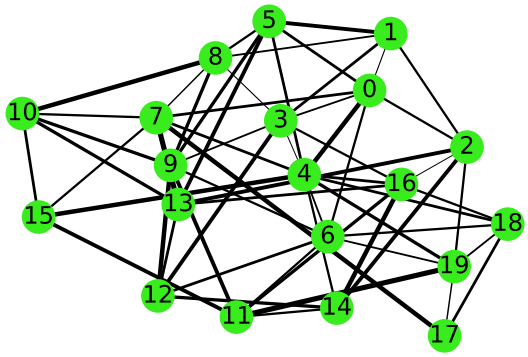
(b) *Grafo 2*



(c) *Grafo 3*



(d) *Grafo 4*



(e) *Grafo 5*

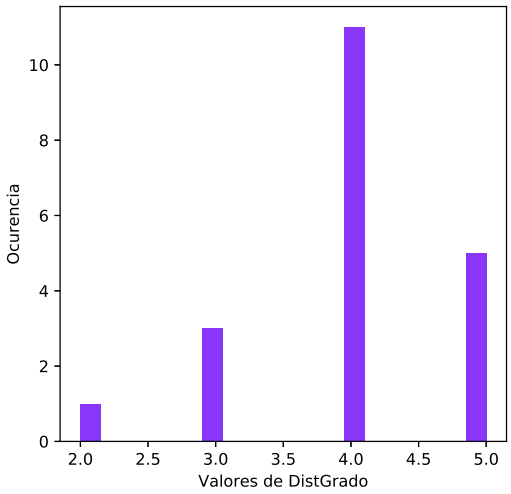
Figura 1: Grafos generados

```
def lee_grafo(nombre):
    ds = pd.read_csv(nombre, header=None)
    G = nx.from_pandas_adjacency(ds)
    return G

def lee_propiedades (nombre):
    ds = pd.read_csv(nombre)
    return ds


i=["grafo1.csv", "grafo2.csv", "grafo3.csv", "grafo4.csv", "grafo5.csv"]
g=[]
for x in i:
    G=lee_grafo(x)

    dic={}
    Nodes=G.nodes;
    dic["Nodo"]=Nodes
    dic["DistGrado"]=[G.degree(i) for i in Nodes]
    dic["CoefAgrup"]=[nx.clustering(G,i) for i in Nodes]
    dic["CentCercania"]=[nx.closeness_centrality(G,i) for i in Nodes]
    dic["CentCarga"]=[nx.load_centrality(G,i) for i in Nodes]
    dic["ExcentCarga"]=[round(nx.eccentricity(G,i), 2) for i in Nodes]
    PageR=nx.pagerank(G, weight="capacity")
    dic["PageR"]=[PageR[i] for i in Nodes]
    df=pd.DataFrame(dic)
    df.to_csv("propiedades"+str(x)+".csv", index=None)
    g.append("propiedades"+str(x)+".csv")

print (g)

j=["DistGrado","CoefAgrup","CentCercania","CentCarga", "ExcentCarga","PageR"]
for y in g:
    H=lee_propiedades(y)
    for i in j:
        fig = plt.figure(figsize=(5, 5))
        ax = fig.add_subplot(1, 1, 1)
        his = ax.hist(H[i],bins=len(H[j]), facecolor='#8a36f8', alpha=0.75)
        ax.set_xlabel("Valores de " + i)
        ax.set_ylabel("Ocurencia")
#         plt.savefig("histograma"+ y + i + ".png", bbox_inches="tight", dpi=100)
        plt.savefig("histograma"+ y + i + ".eps", bbox_inches="tight", dpi=100)
        plt.show()

print("fin")
```

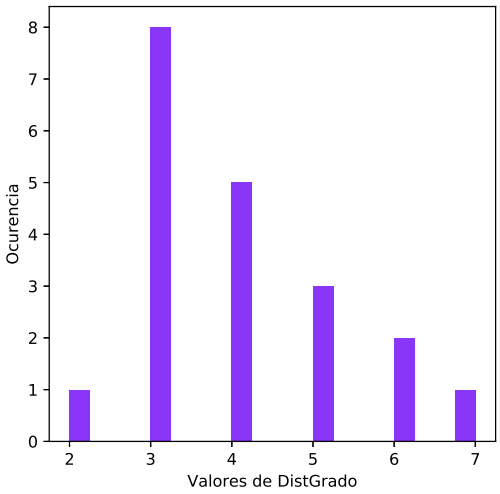
T5histoporopiedad.py

Para tener un avance del comportamiento de estas propiedades de los vértices las cuales se emplearan en el análsis de varianza más adelante, para cada grafo se realizaron los histogramas de cada una de las propiedades (distribución de grado, coeficiente de agrupamiento, centralidad de cercanía, centralidad de carga, excentricidad y *pagerank*) por cada grafo. Los mismos se muestran a continuación:

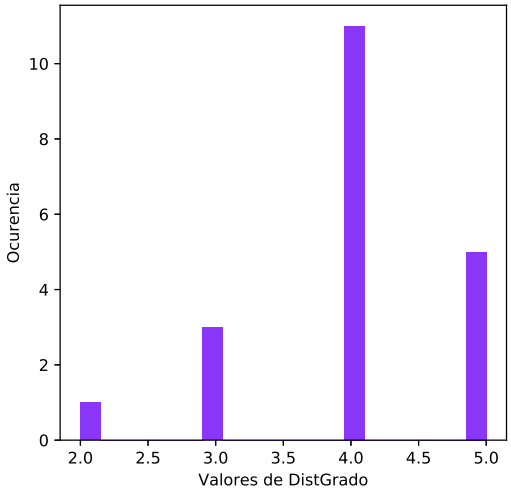
Análisis del comportamiento de la distribución de grado en cada grafo. Ver figura 2 de la página 7



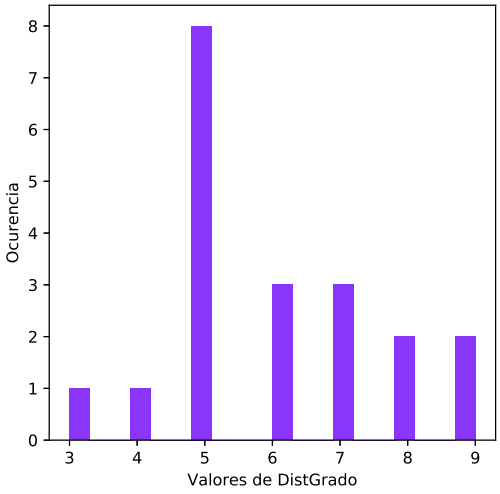
(a) *Grafo 1*



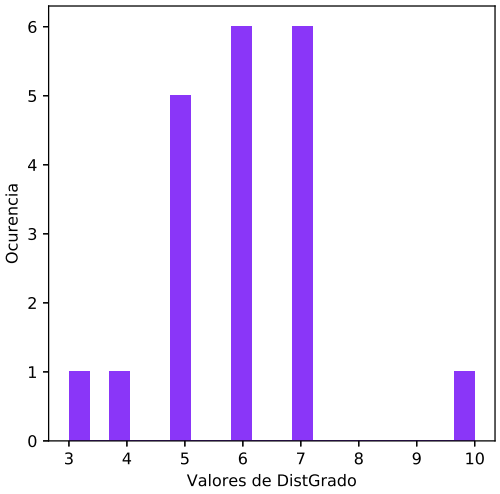
(b) *Grafo 2*



(c) *Grafo 3*



(d) *Grafo 4*



(e) *Grafo 5*

Figura 2: Histogramas de distribución de grado por grafo

Análisis del comportamiento del coeficiente de agrupamiento. Ver figura 3 de la página 9

Análisis del comportamiento de la centralidad de cercanía en cada grafo. Ver figura 4 de la página 10

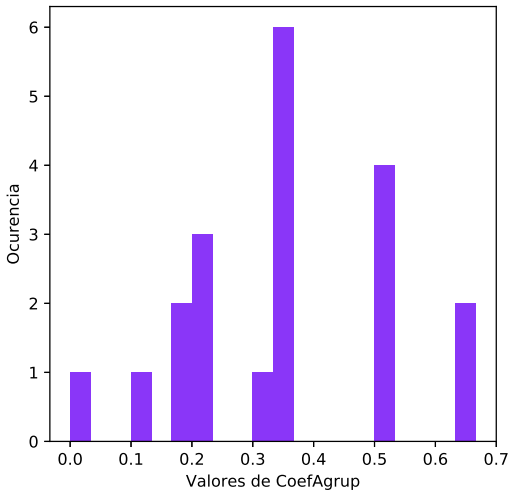
Análisis del comportamiento de la centralidad de carga en cada grafo. Ver figura 5 de la página 11

Análisis del comportamiento de la excentricidad en cada grafo. Ver figura 6 de la página 12

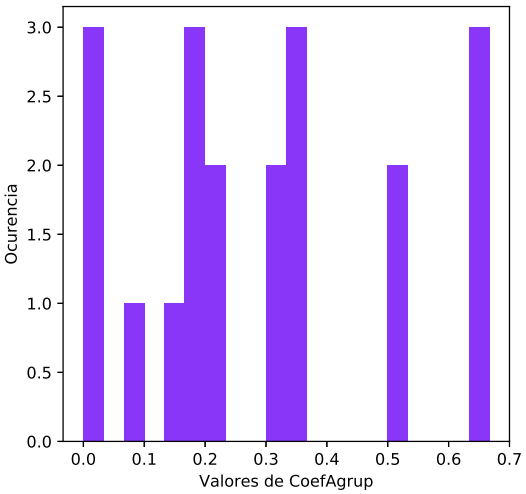
Análisis del comportamiento del *pagerank* en cada grafo. Ver figura 7 de la página 13

Al observar el comportamiento de estas variables se aprecia que no describen una distribución normal, por lo que no es recomendable hacer el análisis de varianza de influencia en el tiempo de ejecución del algoritmo y el flujo máximo usando el promedio, sino la mediana.

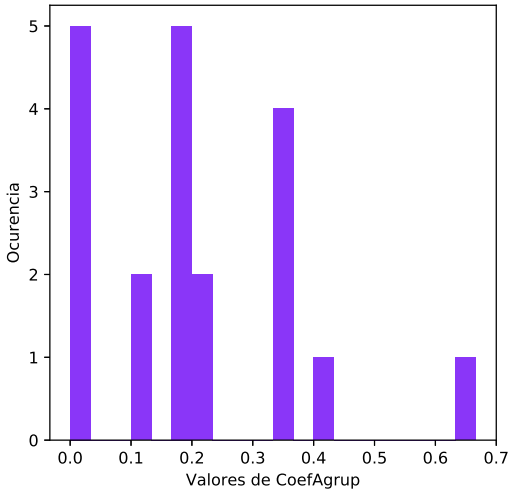
Para realizar el cálculo del tiempo de ejecución del algoritmo y el flujo máximo variando los nodos fuentes y sumideros de modo que toda la información quedase guardada en un .csv que pudiera emplearse en el análisis de varianza (ANOVA)se usó el siguiente código:



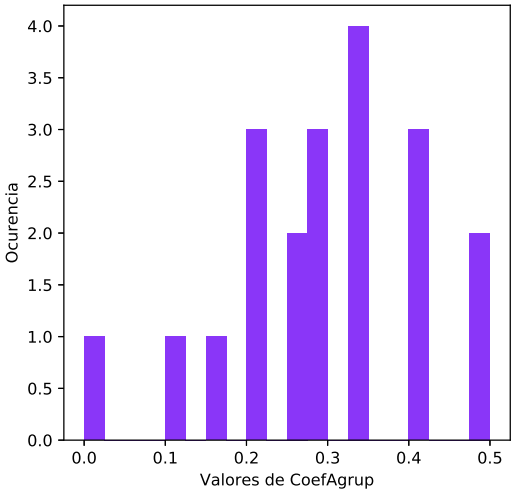
(a) *Grafo 1*



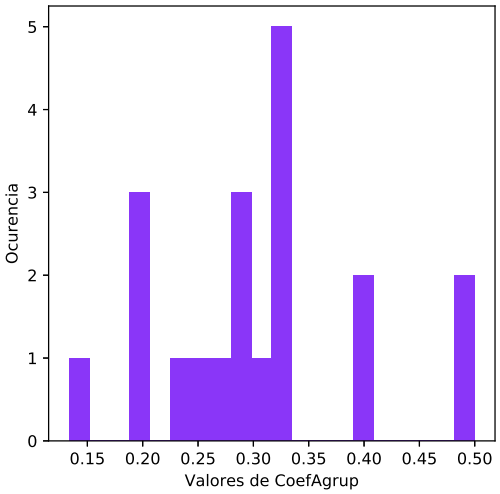
(b) *Grafo 2*



(c) *Grafo 3*

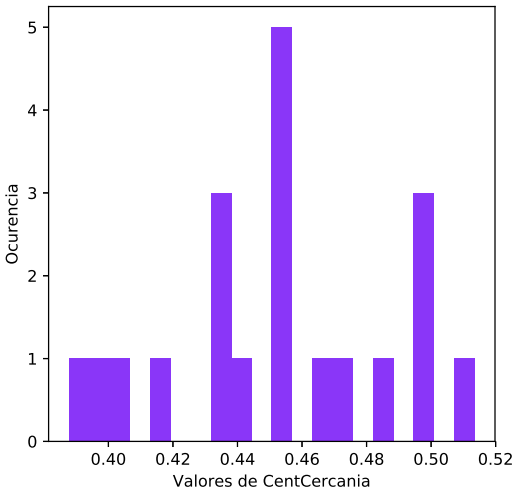


(d) *Grafo 4*

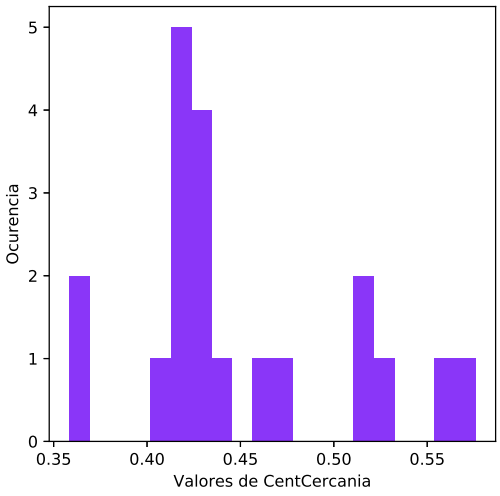


(e) *Grafo 5*

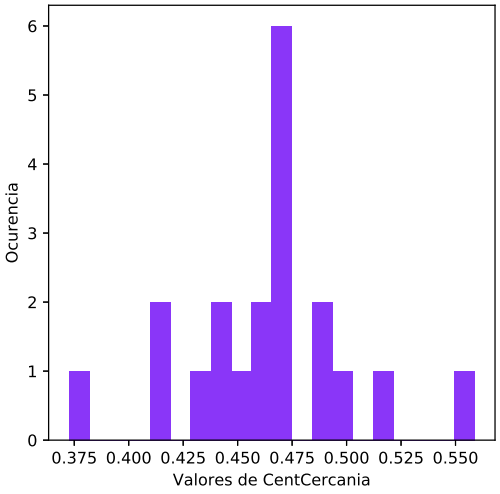
Figura 3: Histogramas del coeficiente de agrupamiento por grafo



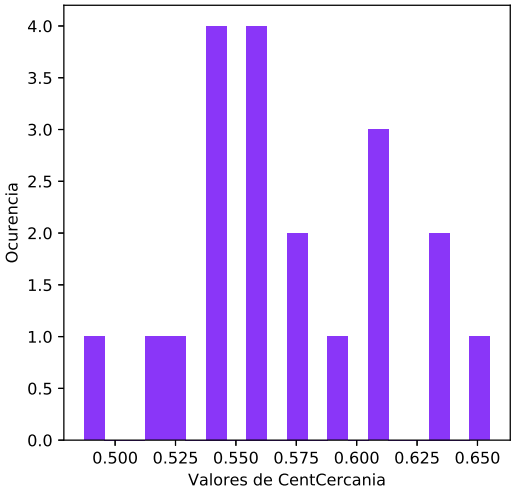
(a) *Grafo 1*



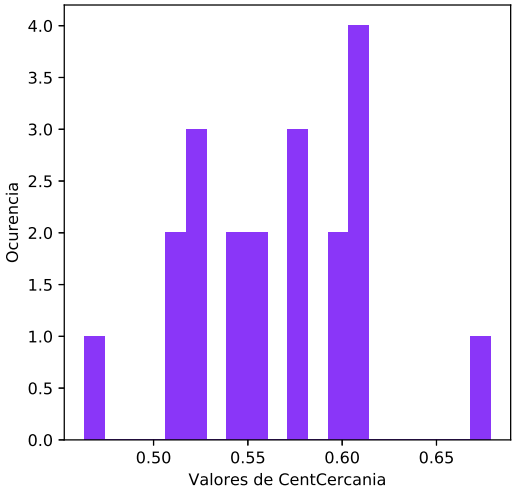
(b) *Grafo 2*



(c) *Grafo 3*

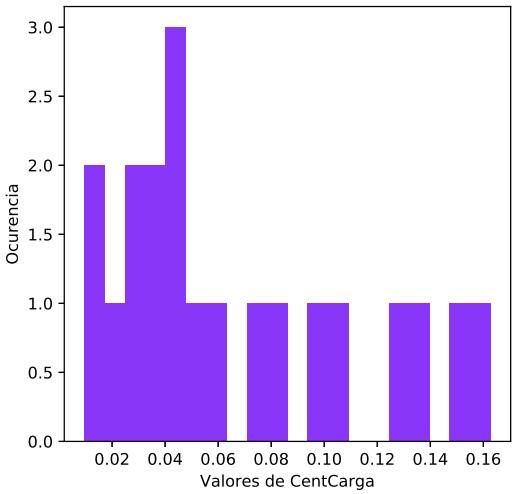


(d) *Grafo 4*

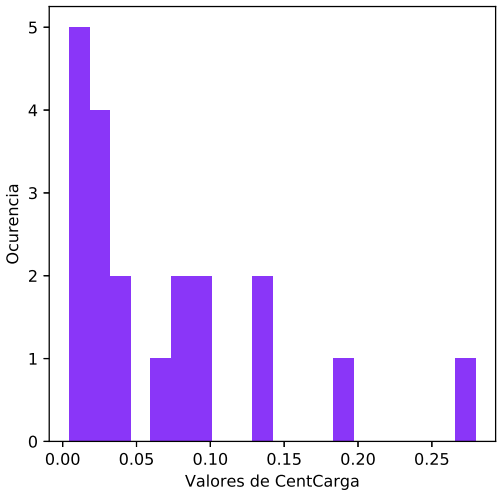


(e) *Grafo 5*

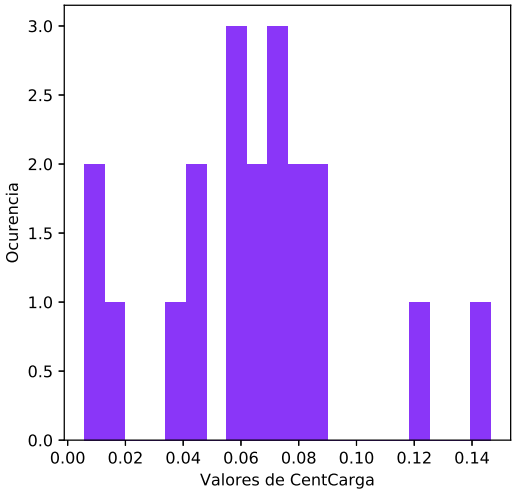
Figura 4: Histogramas de la centralidad de cercanía



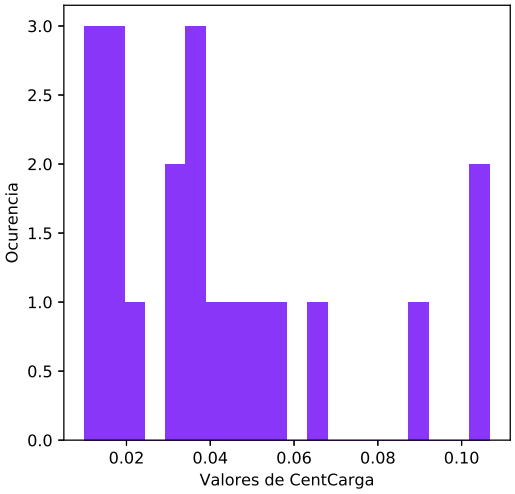
(a) *Grafo 1*



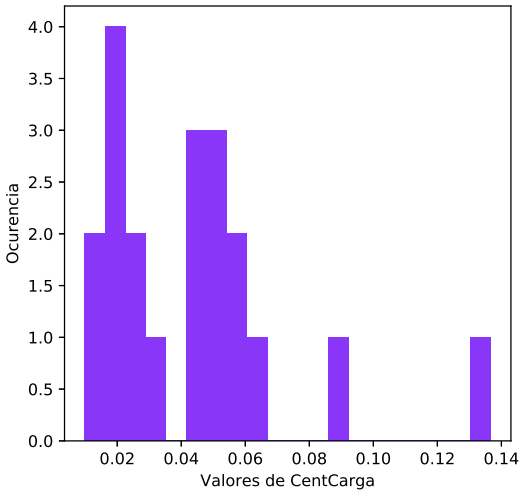
(b) *Grafo 2*



(c) *Grafo 3*

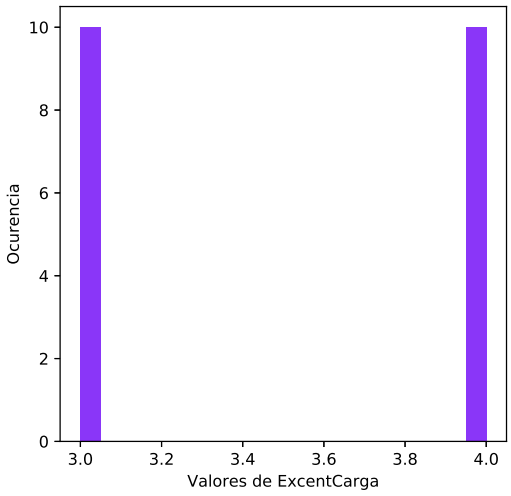


(d) *Grafo 4*

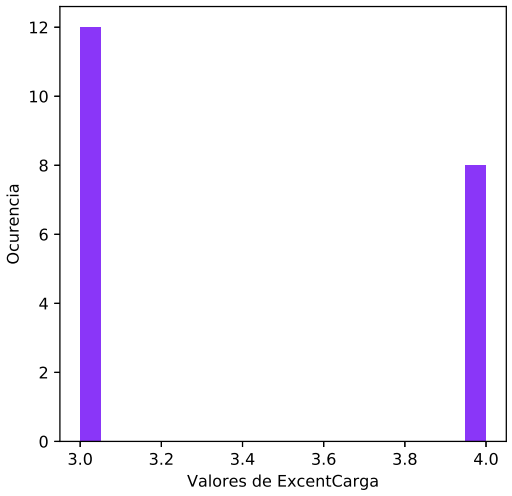


(e) *Grafo 5*

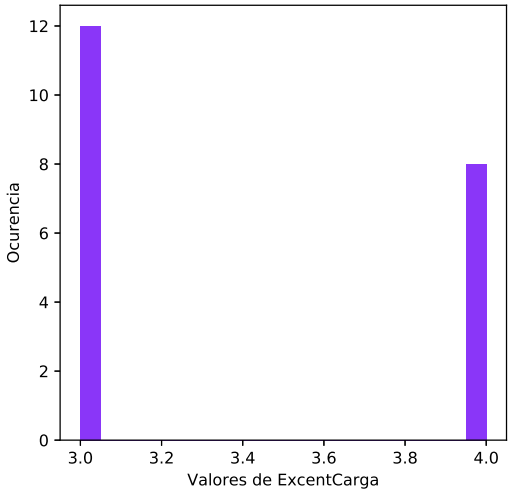
Figura 5: Histogramas de la centralidad de carga



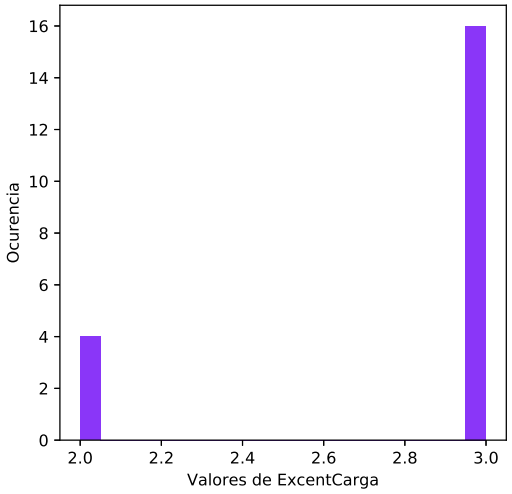
(a) *Grafo 1*



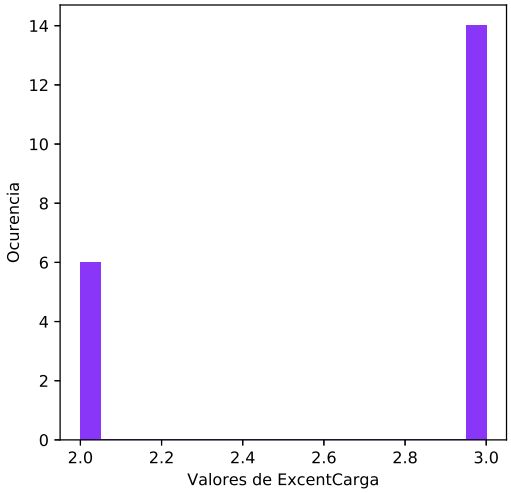
(b) *Grafo 2*



(c) *Grafo 3*

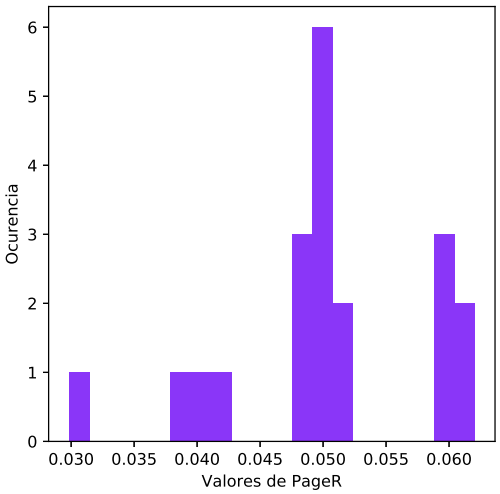


(d) *Grafo 4*

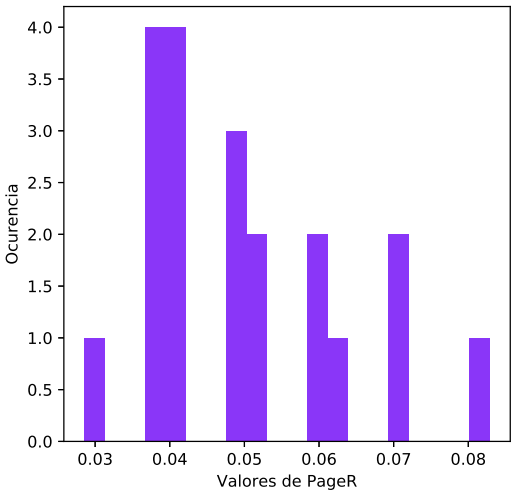


(e) *Grafo 5*

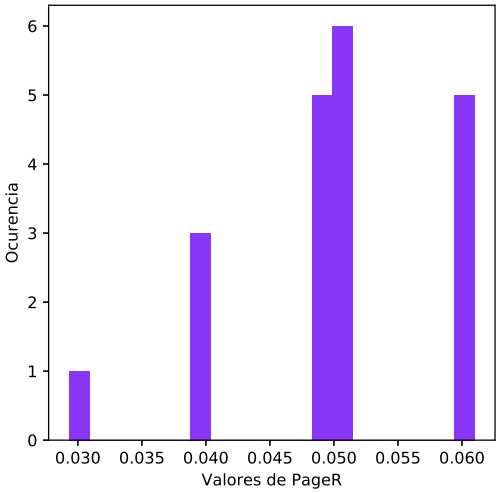
Figura 6: Histogramas de la excentricidad



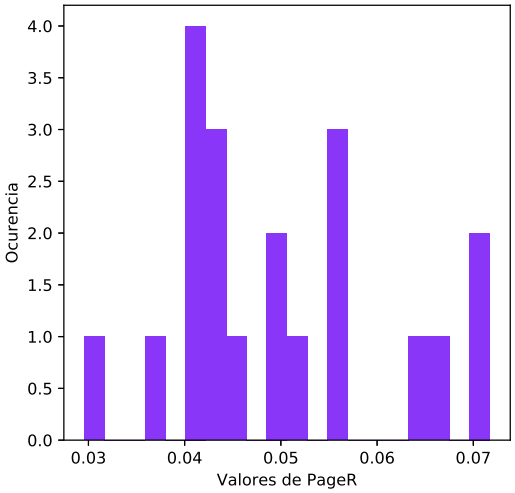
(a) *Grafo 1*



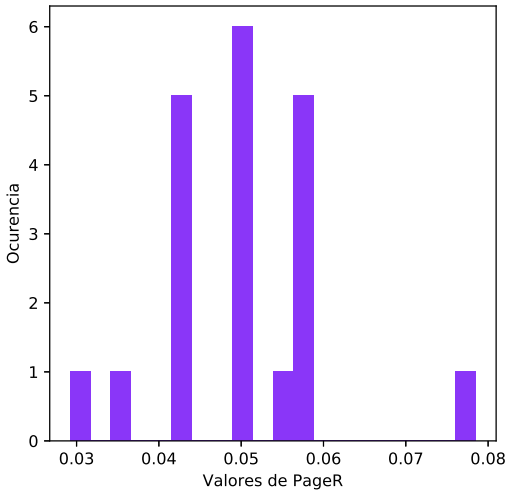
(b) *Grafo 2*



(c) *Grafo 3*



(d) *Grafo 4*



(e) *Grafo 5*

Figura 7: Histogramas del *pagerank*

```
def calcula_tiempo(G,a,b):
    start_time=time()
#    fm=nx.maximum_flow(G, a, b,capacity="weight")
    for i in range (100):
        fm=edmonds_karp(G, a, b,capacity="weight")
        print(i)
    time_elapsed = time() - start_time

    return time_elapsed


def Todo(G,nombre):
    dic={"Grafo":[],
         "Fuente":[],
         "Sumidero":[] ,
         "Mediatime":[],
         "Medianatime":[],
         "Stdtime":[],
         "Flujomaximo":[],
         "DistGrado":[],
         "CoefAgrup":[],
         "CentCercania":[],
         "CentCarga":[],
         "ExcentCarga":[],
         "PageR":[]}

    Nodes=G.nodes;
    for i in Nodes:
        for j in Nodes:
            if i!=j:
                t=[]
                for k in range(10):
                   t.append(calcula_tiempo(G,i,j))
                dic["Grafo"].append(nombre)
                dic["Fuente"].append(i)
                dic["Sumidero"] .append(j)
                dic["Mediatime"].append(np.mean(t))
                dic["Medianatime"].append(np.median(t))
                dic["Stdtime"].append(np.std(t))
                dic["Flujomaximo"].append(nx.maximum_flow_value(G,i,j,capacity="
    weight"))

                dic["DistGrado"].append(G.degree(i))
                dic["CoefAgrup"].append(nx.clustering(G,i))
                dic["CentCercania"].append(nx.closeness_centrality(G,i))
                dic["CentCarga"].append(nx.load_centrality(G,i))
                dic["ExcentCarga"].append(nx.eccentricity(G,i))
                PageR=nx.pagerank(G,weight="capacity")
                dic["PageR"].append(PageR[i])


    df=pd.DataFrame(dic)
    df.to_csv("CSVunido.csv", index=None, mode="a")


i=["grafo1", "grafo2", "grafo3", "grafo4", "grafo5"]
for x in i:
```

```
57      print (x)
58      G=lee_grafo(x)
59      Todo(G,x)
```

T5Paraunircsvfinal.py

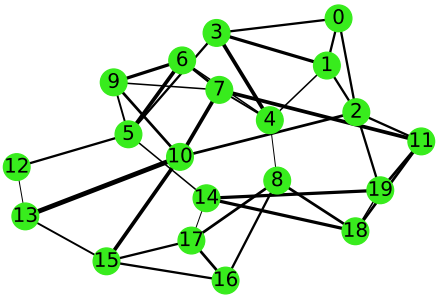
.

3. Influencia en el óptimo al variar nodos fuentes y sumideros.

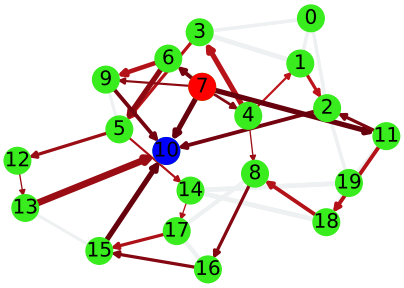
Al variar los nodos fuentes y sumideros fue calculado el valor de flujo máximo que se obtenía en cada caso, evidenciándose que existe variación en el óptimo para cada uno de los cinco grafos. A continuación se muestra para cada uno de los cinco grafos generados inicialmente el grafo inicial con la propuesta de mejor y peor fuente y sumidero. La fuente aparece con color rojo y el sumidero con color azul. De igual modo se emplea el degradado de color rojo para representar con rojo más intenso la mayor cantidad de flujo y más claro la menor cantidad de flujo que pasa por las aristas de la red residual que devuelve el algoritmo de flujo máximo seleccionado.

En la figura 8 de la página 16 se observa la variación realizada para el grafo uno, en la figura 9 de la página 17 la variación del grafo dos, en la figura 10 de la página 18 la variación del grafo tres, en la figura 11 de la página 19 la variación del grafo cuatro y figura 12 de la página 20 la variación del grafo cinco.

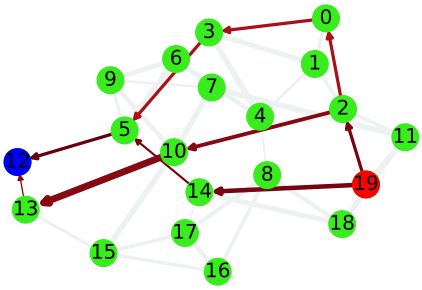
Seguidamente se muestra el código empleado en esta sección



(a) *Sin flujo*

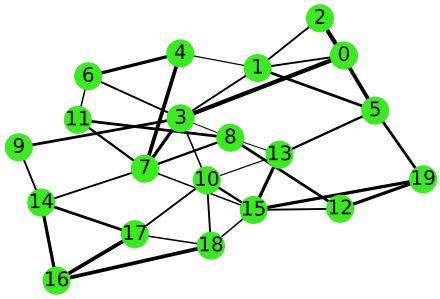


(b) *Flujo máximo 59u*

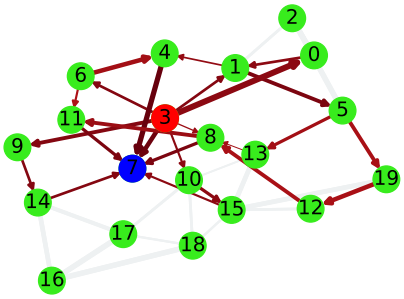


(c) *Flujo mínimo 18u*

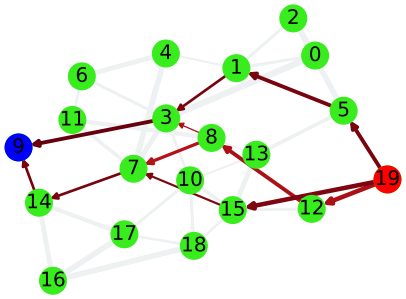
Figura 8: Variación de fuente y sumidero en grafo 1



(a) *Sin flujo*

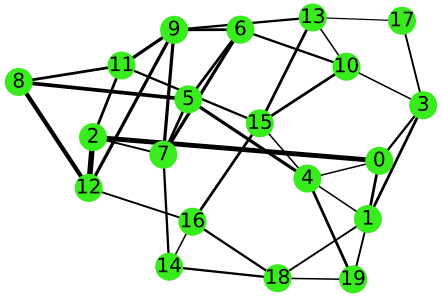


(b) *Flujo máximo 68u*

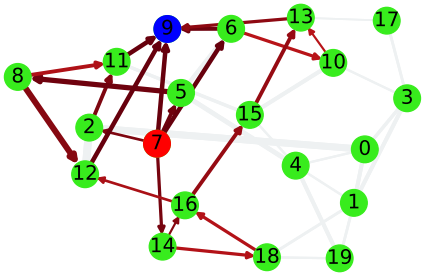


(c) *Flujo mínimo 22u*

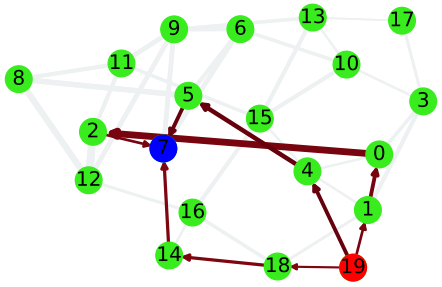
Figura 9: Variación de fuente y sumidero en grafo 2



(a) *Sin flujo*

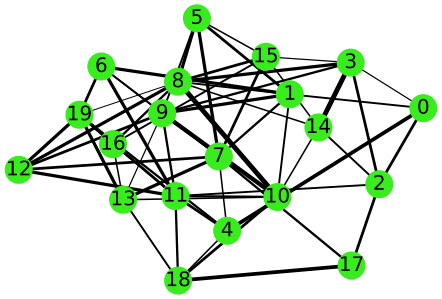


(b) *Flujo máximo 59u*

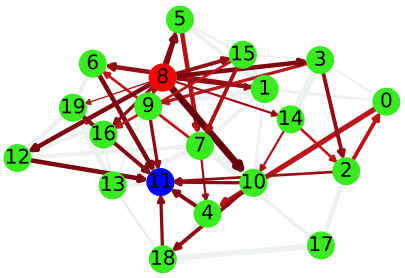


(c) *Flujo mínimo 18u*

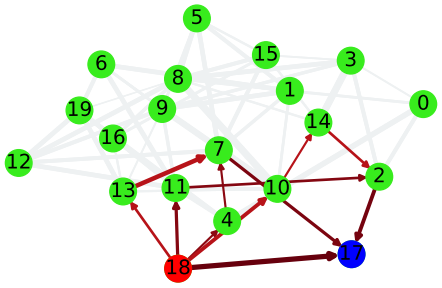
Figura 10: Variación de fuente y sumidero en grafo 3



(a) *Sin flujo*

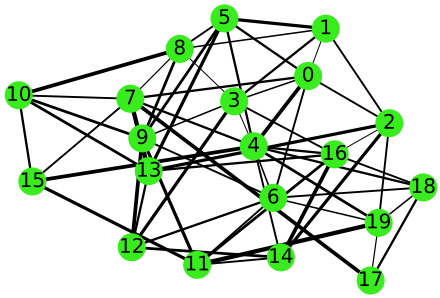


(b) *Flujo máximo 102u*

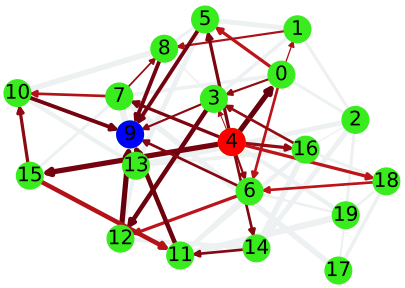


(c) *Flujo mínimo 37u*

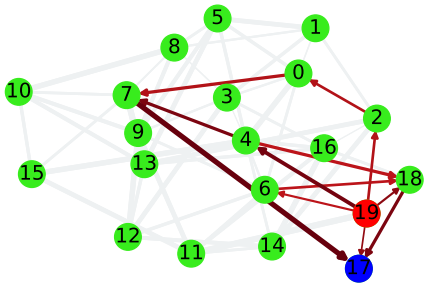
Figura 11: Variación de fuente y sumidero en grafo 4



(a) *Sin flujo*



(b) *Flujo máximo 82u*



(c) *Flujo mínimo 33u*

Figura 12: Variación de fuente y sumidero en grafo 5

```
def Todo(G):
    dic={"Fuente":[],
         "Sumidero":[]  ,
         "Mediatime":[],
         "Medianatime":[],
         "Stdtime":[],
         "Flujomaximo":[],
         "DistGrado":[],
         "CoefAgrup":[],
         "CentCercania":[],
         "CentCarga":[],
         "ExcentCarga":[],
         "PageR":[]}

    Nodes=G.nodes;
    for i in Nodes:
        for j in Nodes:
            if i!=j:
                t=[]
                for k in range(10):
                   t.append(calcula_tiempo(G,i,j))
                dic["Fuente"].append(i)
                dic["Sumidero"] .append(j)
                dic["Mediatime"].append(np.mean(t))
                dic["Medianatime"].append(np.median(t))
                dic["Stdtime"].append(np.std(t))
                dic["Flujomaximo"].append(nx.maximum_flow_value(G,i,j,capacity="
    weight"))

                dic["DistGrado"].append(G.degree(i))
                dic["CoefAgrup"].append(nx.clustering(G,i))
                dic["CentCercania"].append(nx.closeness_centrality(G,i))
                dic["CentCarga"].append(nx.load_centrality(G,i))
                dic["ExcentCarga"].append(nx.eccentricity(G,i))
                PageR=nx.pagerank(G,weight="capacity")
                dic["PageR"].append(PageR[i])


    df=pd.DataFrame(dic)
    df.to_csv("Todosvaloresg5.csv", index=None)

def DibujarRes(G,pos,fuentes,sumideros):
#     pos=nx.spring_layout(R)
    sinflujo=[]
    flowconflujo=[]
    conflujo=[]
    sinflujopesos=[]
    conflujopesos=[]
    maxi=0
    for edge in G.edges():
        if G.edges[edge]['flow']==0:
           sinflujo.append(edge)
           sinflujopesos.append( G.edges[edge]['capacity'] )
        elif G.edges[edge]['flow']>0:
           conflujo.append(edge)
           conflujopesos.append( G.edges[edge]['capacity'] )
           flowconflujo.append(G.edges[edge]['flow'] )
```

```
57      flowconflujo[:] = [x+50 for x in flowconflujo]
58      for i in flowconflujo:
59          if i>maxi:
60              maxi=i
61      sinflujopesos[:] = [x/10*x/5 for x in  sinflujopesos]
62      conflujopesos[:] = [x/10*x/5 for x in  conflujopesos]
63      nx.draw_networkx_nodes(G, pos, node_size=400, node_color='#38ec1d', node_shape='
        o')
64      nx.draw_networkx_nodes(G, pos,nodelist=fuentes, node_size=400, node_color='r',
        node_shape='o')
65      nx.draw_networkx_nodes(G, pos,nodelist=sumideros, node_size=400, node_color='b',
         node_shape='o')
66      nx.draw_networkx_edges(G, pos, edgelist=sinflujo, edge_color='#eef1f2', width=
        sinflujopesos,arrows=False)
67      nx.draw_networkx_edges(G, pos,edgelist=conflujo, edge_cmap=plt.cm.Reds, width=
        conflujopesos,edge_color=flowconflujo,edge_vmin=0,edge_vmax=maxi)
68      labels = {}
69      for i in G.nodes:
70          labels[i]=str(i)
71      nx.draw_networkx_labels(G, pos, labels, font_size=15 )
72      plt.axis('off')
73      plt.savefig("fig5b1.png",dpi=600)
74      plt.savefig("fig5b1.eps",dpi=600)
75
76
77  G=lee_grafo("grafo5.csv")
78  GeneraPropiedades(G)
79  DibujaGrafo(G)
80  Todo(G)
81
82  R=edmonds_karp(G,4,9,capacity="weight")
83  pos = pd.read_csv("pos_grafo5.csv", header=None)
84  DibujarRes(R,pos,[4],[9])
```

T5ProTimeFLujo.py

.

4. Análisis de varianza (ANOVA) y prueba de *Tukey* para determinar la relación entre las propiedades de los nodos y las variable dependientes estudiadas.

Para realizar el análisis del comportamiento de la variables dependientes *tiempo de ejecución* y *flujo máximo* con respecto a cada factor a analizar se realizó un análisis de varianza (ANOVA) para cada factor.

En el caso de las propieddes que se usan como factores en el ANOVA se realizó un histograma para cada una de ellas de modo que se puediera llevar a escalas los valores de las mismas.

Los histogramas se emplean para establecer los rangos de valores asignándoles una etiqueta de bajo, medio y alto para cada propiedad.

El resultado del análisis ANOVA con respecto al tiempo de ejecución para cada propiedad se muestra en los cuadros del uno al seis.Según los resultados en todas las propiedas se rechaza la hipótesis nula por lo que todas influyen en el tiempo excepto la centralidad de carga de rango alto, en la que según los resultados obtenidos se acepta la hipótesis nula. Esto se puede corroborar al analizar los diagramas de cajas de cada una de las propiedades. Ver figura 13 de la página 24. Del análisis de los resultados se aprecia que el que menos influye es la centralidad de carga, como se observa también en los diagramas de caja y prueba de *Tukey* (ver figura 14 de la página 25) el resto de las propiedades sí influyen en el tiempo de ejecución del algoritmo al variar los nodos fuentes y sumideros.

Cuadro 1: ANOVA Centralidad de Carga

Source	**SS**	**DF**	**MS**	**F**	**p-unc**	**np2**
CentCarga	0	2	0	0.058	0.94360248	0
Within	0.126	1897	0	-	-	-

Cuadro 2: ANOVA Centralidad de cercanía

Source	**SS**	**DF**	**MS**	**F**	**p-unc**	**np2**
CentCercania	0.024	2	0.012	221.416	3.76E-87	0.189
Within	0.102	1897	0	-	-	-

Cuadro 3: ANOVA Coeficiente de agrupamiento

Source	**SS**	**DF**	**MS**	**F**	**p-unc**	**np2**
CoefAgrup	0.007	2	0.004	56.285	1.79E-24	0.056
Within	0.119	1897	0	-	-	-

Los resultados *Tukey* de este análisis se muestran en la figura 14 de la página 25

Al realizar el análisis ANOVA para verificar la influencia de las propiedades en el flujo máximo se obtuvieron los resultados que se muestran en los cuadros del 7 al 12. En los mismos se aprecia que se rechaza la hipótesis nula en todos los casos excepto en la propiedad de centralidad de carga en la que se acepta la hipótesis nula. Se incluyen los diagramas de caja (ver figura 15 de la página 27) y las pruebas de *Tukey* para profundizar en los resultados (ver figura 16 de la página 28). Se observa que la propiedad que menos influye es la centralidad de carga para valores altos, el resto de las propiedades sí influye al variar los nodos fuestes y sumideros en el valor de flujo máximo.

Referencias

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

[2] Desarrolladores de Networkx. Add-edge. `https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.MultiDiGraph.add_edge.html`. Accessed:2019-04-19.

Cuadro 4: ANOVA Distribución de Grado

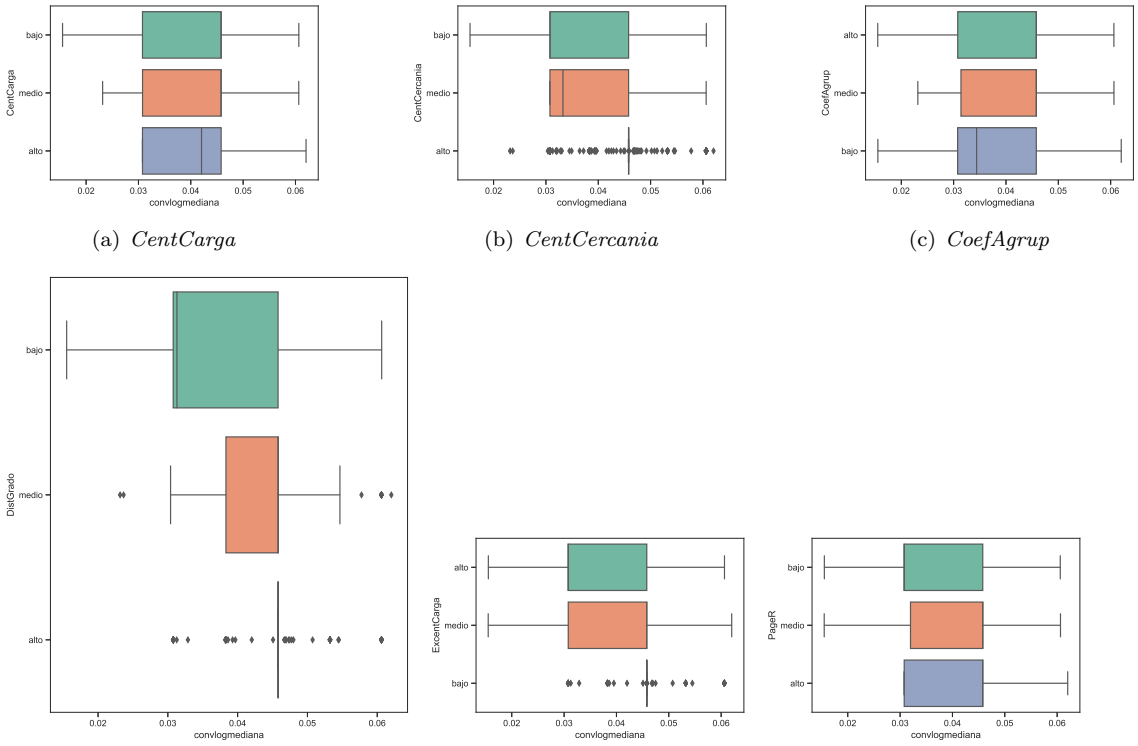
Source	SS	DF	MS	F	p-unc	np2
DistGrado	0.023	2	0.011	209.861	4.61E-83	0.181
Within	0.103	1897	0	-	-	-

Cuadro 5: ANOVA Excentricidad

Source	SS	DF	MS	F	p-unc	np2
ExcentCarga	0.013	2	0.006	104.776	6.90E-44	0.099
Within	0.113	1897	0	-	-	-

Cuadro 6: ANOVA *PageRange*

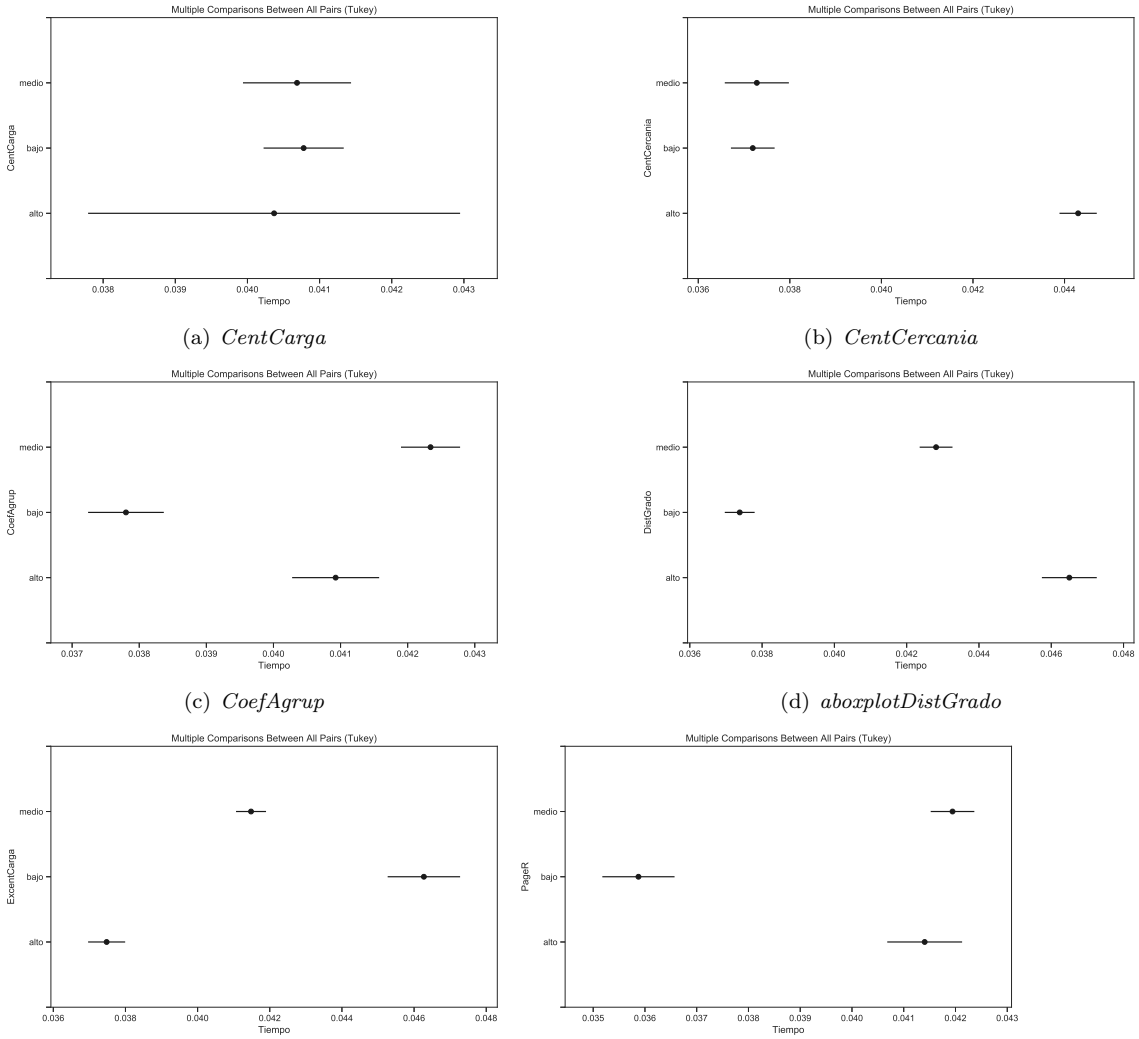
Source	SS	DF	MS	F	p-unc	np2
PageR	0.01	2	0.005	82.216	5.72E-35	0.08
Within	0.116	1897	0	-	-	-



(a) *CentCarga* (b) *CentCercania* (c) *CoefAgrup*

(d) *aboxplotDistGrado* (e) *Excent* (f) *PageR*

Figura 13: Diagrama de caja del ANOVA con Tiempo



(a) *CentCarga* (b) *CentCercania*

(c) *CoefAgrup* (d) *aboxplotDistGrado*

(e) *Excent* (f) *PageR*

Figura 14: Diagrama de TUKEY para ANOVA con Tiempo

Cuadro 7: ANOVA flujo max CentCarga

Source	**SS**	**DF**	**MS**	**F**	**p-unc**	**np2**
CentCarga	0.471	2	0.236	2.473	0.08461665	0.003
Within	180.834	1897	0.095	-	-	-

Cuadro 8: ANOVA flujo max CentCercanía

Source	SS	DF	MS	F	p-unc	np2
CentCercania	68.685	2	34.343	578.476	7.16E-197	0.379
Within	112.62	1897	0.059	-	-	-

Cuadro 9: ANOVA flujo max CoefAgrup

Source	SS	DF	MS	F	p-unc	np2
CoefAgrup	19.784	2	9.892	116.181	2.53E-48	0.109
Within	161.521	1897	0.085	-	-	-

Cuadro 10: ANOVA flujo max DistGrado

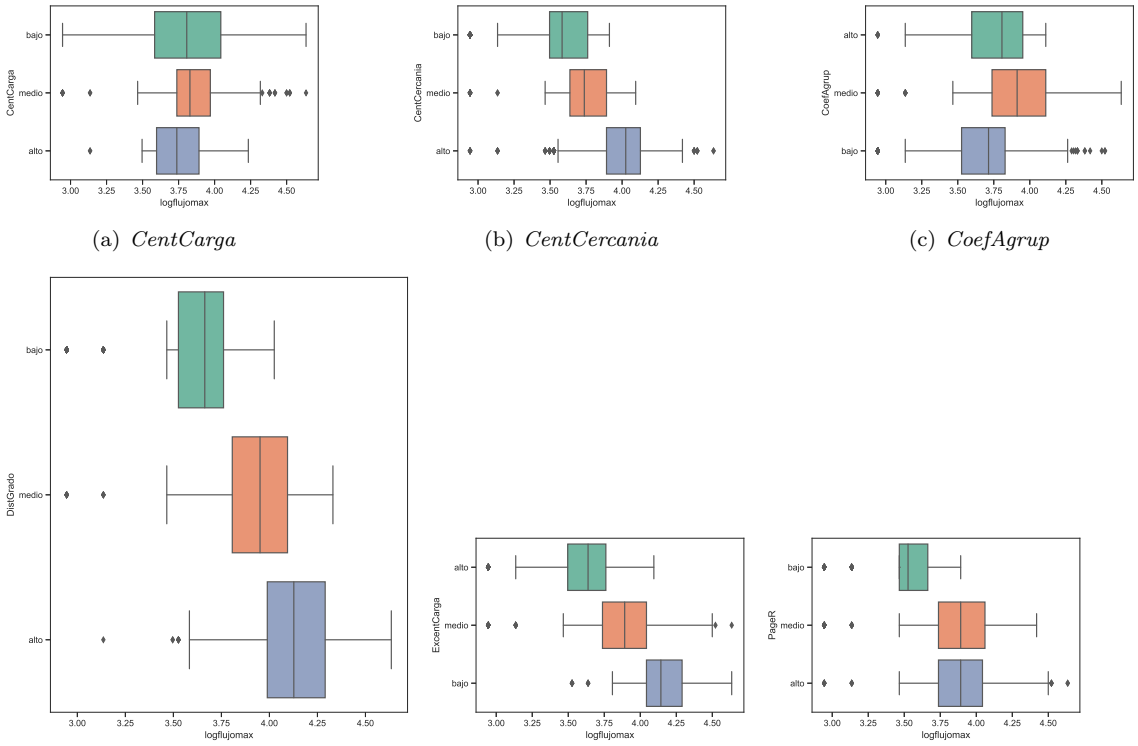
Source	SS	DF	MS	F	p-unc	np2
DistGrado	68.851	2	34.426	580.732	1.77E-197	0.38
Within	112.454	1897	0.059	-	-	-

Cuadro 11: ANOVA flujo max Excent

Source	SS	DF	MS	F	p-unc	np2
ExcentCarga	54.856	2	27.428	411.472	3.69E-149	0.303
Within	126.45	1897	0.067	-	-	-

Cuadro 12: ANOVA flujo max *PageR*

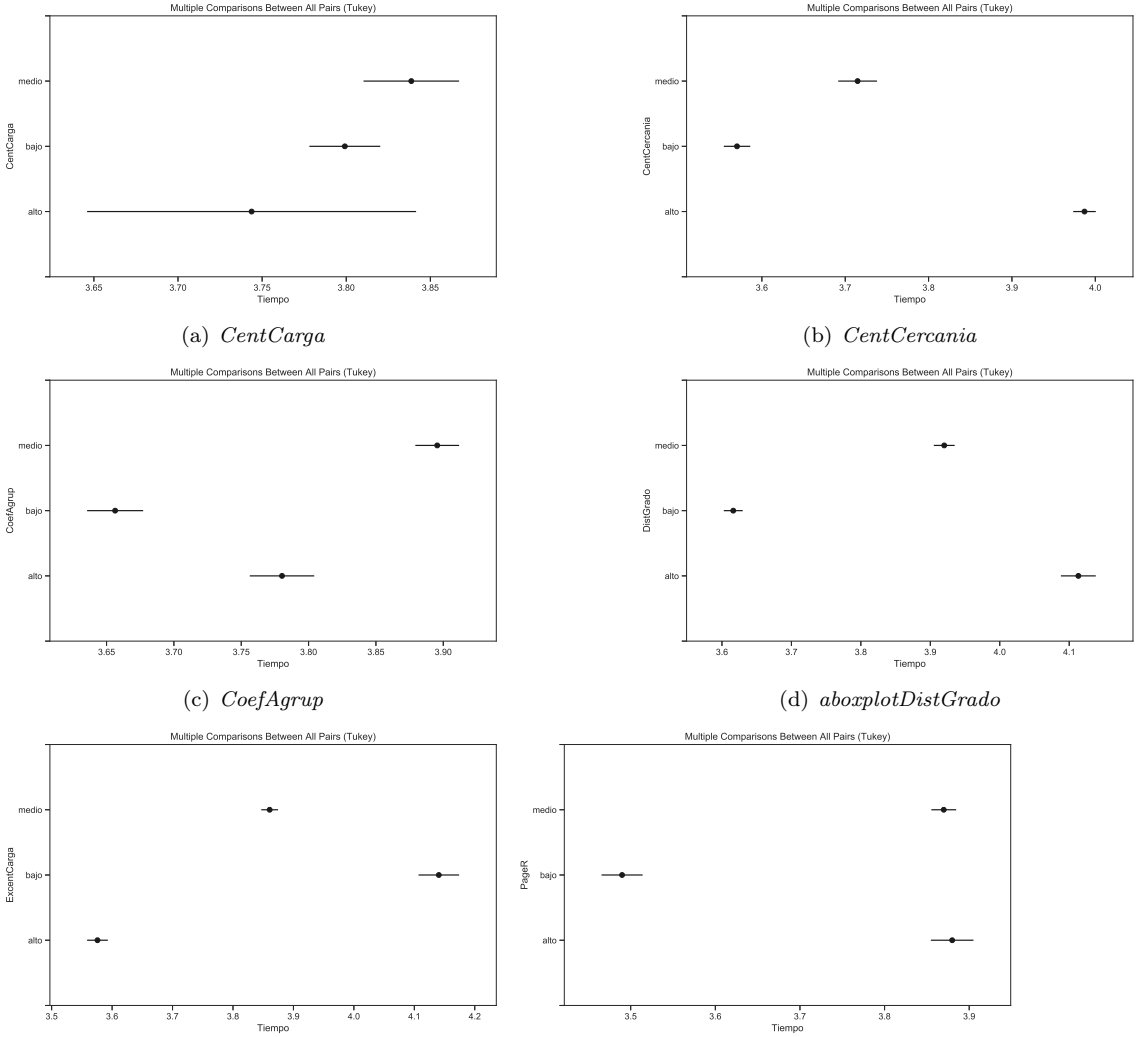
Source	SS	DF	MS	F	p-unc	np2
PageR	40.988	2	20.494	277.069	2.70E-106	0.226
Within	140.317	1897	0.074	-	-	-



(a) *CentCarga* (b) *CentCercania* (c) *CoefAgrup*

(d) *aboxplotDistGrado* (e) *Excent* (f) *PageR*

Figura 15: Diagrama de caja del ANOVA con flujo máximo



(a) *CentCarga*

(b) *CentCercania*

(c) *CoefAgrup*

(d) *aboxplotDistGrado*

(e) *Excent*

(f) *PageR*

Figura 16: Diagrama de TUKEY para ANOVA con flujo máximo

[3] Desarrolladores de Scipy.Org. Función estadística *scipy.stats.truncnorm*. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.truncnorm.html. Accessed:2019-04-19.

[4] 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-CollectiveDynamicsOfSmallWorldNetworks.pdf.

[5] Pedro A. Solares Hernández. *Redes aleatorias de pequeño mundo y libres de escalas*. Universidad Politécnica de Valencia, 2017. Accessed:2019-04-26.

[6] Valerie De la Cruz. ¿what are the applications of random graphs? develop python with pycharm. https://www.quora.com/What-are-the-applications-of-random-graphs, 2018.

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

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