Tarea 3 Optimización de Flujo de Redes

1985274

19 de marzo de 2019

En este trabajo se presenta un análisis de varios algoritmos selecionados para analizar su comportamiento sobre algunos grafos seleccionados. Para esto se ha utilizado el lenguaje **Python** en su versión 3.7 [5], 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 [2], **Matplotlib** [1], **Numpy** [3], así como la librería **Panda** [4].

1. Histograma 1

Se utilizaron los histogramas porque permiten visualizar la distribución y dispersión de los datos. El eje horizontal está formado por los valores del tiempo de ejecución en segundos, mientras que en el eje vertical se representan los datos de probabilidad de ocurrencia. En este caso particular los histogramas muestran la frecuencia de ocurrencia en un intervalo de tiempo determinado.

Los grafos utilizados comprenden instancias de 150, 301, 601, 801 y 1001 nodos respectivamente y para cada uno se corrieron los cinco algoritmos, con el fin de obtener 25 combinaciones. Estos grafos utilizados están conformados por 11325, 45150, 180300, 320400 y 500500 arcos respectivamente. Se ejecutaron las corridas 50 veces, luego se ejecutaron otras 30 veces adicionales y se registraron los tiempos que se demoraron las últimas 30 corridas. Los algoritmos se ejecutaron sobre 5 tipos de grafos simples cíclicos y acíclicos, en ambos casos con pesos en los arcos.

En la figura 1 se ilustan cinco histogramas, uno por cada algoritmo utilizado que son **betweenness centrality**, **minimum spanning tree**, **greedy color**, el algoritmo **max clique** y el algoritmo **maximal matching**, en ese orden en que han sido mencionados.

El histograma del algoritmo uno que es el **betweenness centrality** refleja que el comportamiento de los tiempos registrados no sigue una distribución normal, o sea no tiene una distribución simétrica.

El histograma del algoritmo dos que es el **minimum spanning tree** refleja que el comportamiento del tiempo registrado no sigue una distribución normal, no tiene una distribución simétrica.

Algoritmos	Nodos	Media
betweenness	150	0.7120185
betweenness	301	5.7420185
betweenness	601	32.542511
betweenness	801	82.115157
betweenness	1001	168.1961
spanning tree	150	0.0336394
spanning tree	301	0.1474
spanning tree	601	0.63160
spanning tree	801	1.165445
spanning tree	1001	1.9261
greedy color	150	0.002992
greedy color	301	0.04987
greedy color	601	0.4987
greedy color	801	0.08979
greedy color	1001	0.142054
max clique	150	5.669975
max clique	301	0.055175
max clique	601	0.097799
max clique	801	1.4071
max clique	1001	0.007919
maximal matching	150	0.00410906
maximal matching	301	0.017745
maximal matching	601	0.07384
maximal matching	801	0.1276
maximal matching	1001	0.2004736

Cuadro 1: Tabla de valores capturados

1.1. Código

```
import datetime
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
tiempos = []
def Generator(num, nameGraf):
    G1 = nx.Graph()
    nodes = []
    edges = []
    for i in range(num):
            nodes.append(i)
    for i in nodes:
         idx = nodes.index(i) + 1
         for j in nodes[idx:len(nodes)]:
             edg = rnd.randint(1,4)
```

```
            if edg:
                G1.add_edge(i,j, distance=rnd.randint(1,10))
                G1.add_edge(j, i, distance=rnd.randint(1,10))
    df = pd.DataFrame()
    df = nx.to_pandas_adjacency(G1, dtype=int, weight="distance")
    df.to_csv(nameGraf, index=None, header=None)
for i in range(1):
    A=Generator(rnd.randint(100,150),"grafos"+str(i)+".csv")

def betweenness_centrality(nombre):
    df = pd.DataFrame()
    df = pd.read_csv(nombre, header=None)
    a = nx.from_pandas_adjacency(df, create_using=nx.Graph())

    for i in range(30):
        inicio = datetime.datetime.now()
        for key in range(50):
            nx.betweenness_centrality(a)
        final = datetime.datetime.now()
        tiempos.append((final - inicio).total_seconds())

    media=np.mean(tiempos)
    desv=np.std(tiempos)
    mediana=np.median(tiempos)
    nodos=nx.number_of_nodes(a)
    arcos=nx.number_of_edges(a)
    salvar=[]
    salvar.append(media)
    salvar.append(desv)
    salvar.append(mediana)
    salvar.append(nodos)
    salvar.append(arcos)
    return salvar
def minimum_spanning_tree(nombre):
    df = pd.read_csv(nombre, header=None)
    b = nx.from_pandas_adjacency(df, create_using=nx.Graph())
    for i in range(30):
        inicio = datetime.datetime.now()
        for key in range(50):
            nx.minimum_spanning_tree(b)
        final = datetime.datetime.now()
        tiempos.append((final - inicio).total_seconds())
    media=np.mean(tiempos)
    desv=np.std(tiempos)
    mediana=np.median(tiempos)
    nodos=nx.number_of_nodes(b)
    arcos=nx.number_of_edges(b)
    salvar=[]
    salvar.append(media)
    salvar.append(desv)
```

```
    salvar.append(mediana)
    salvar.append(nodos)
    salvar.append(arcos)
    return salvar

def greedy_color(nombre):
    df = pd.read_csv(nombre, header=None)
    b = nx.from_pandas_adjacency(df, create_using=nx.Graph())
    for i in range(30):
        inicio = datetime.datetime.now()
        for key in range(50):
            nx.greedy_color(b)
        final = datetime.datetime.now()
        tiempos.append((final - inicio).total_seconds())
    media=np.mean(tiempos)
    desv=np.std(tiempos)
    mediana=np.median(tiempos)
    nodos=nx.number_of_nodes(b)
    arcos=nx.number_of_edges(b)
    salvar=[]
    salvar.append(media)
    salvar.append(desv)
    salvar.append(mediana)
    salvar.append(nodos)
    salvar.append(arcos)#
    return salvar

def max_clique(nombre):
    df = pd.read_csv(nombre, header=None)
    b = nx.from_pandas_adjacency(df, create_using=nx.Graph())
    for i in range(30):
        inicio = datetime.datetime.now()
        for key in range(50):
            nx.make_max_clique_graph(b, create_using=None)
        final = datetime.datetime.now()
        tiempos.append((final - inicio).total_seconds())
    media=np.mean(tiempos)
    desv=np.std(tiempos)
    mediana=np.median(tiempos)
    nodos=nx.number_of_nodes(b)
    arcos=nx.number_of_edges(b)
    salvar=[]
    salvar.append(media)
    salvar.append(desv)
    salvar.append(mediana)
    salvar.append(nodos)
    salvar.append(arcos)
    return salvar

def maximal_matching(nombre):
```

```
116     df = pd.read_csv(nombre, header=None)
117     b = nx.from_pandas_adjacency(df, create_using=nx.Graph())
118     for i in range(30):
119         inicio = datetime.datetime.now()
120         for key in range(50):
121             nx.maximal_matching(b)
122         final = datetime.datetime.now()
123         tiempos.append((final - inicio).total_seconds())
124     media=np.mean(tiempos)
125     desv=np.std(tiempos)
126     mediana=np.median(tiempos)
127     nodos=nx.number_of_nodes(b)
128     arcos=nx.number_of_edges(b)
129     salvar=[]
130     salvar.append(media)
131     salvar.append(desv)
132     salvar.append(mediana)
133     salvar.append(nodos)
134     salvar.append(arcos)
135     return salvar
136 
137 def guardarDatosenCSV(nombre):
138     valores={'Alg1':[],
139              'grafo':[],
140              'media':[],
```

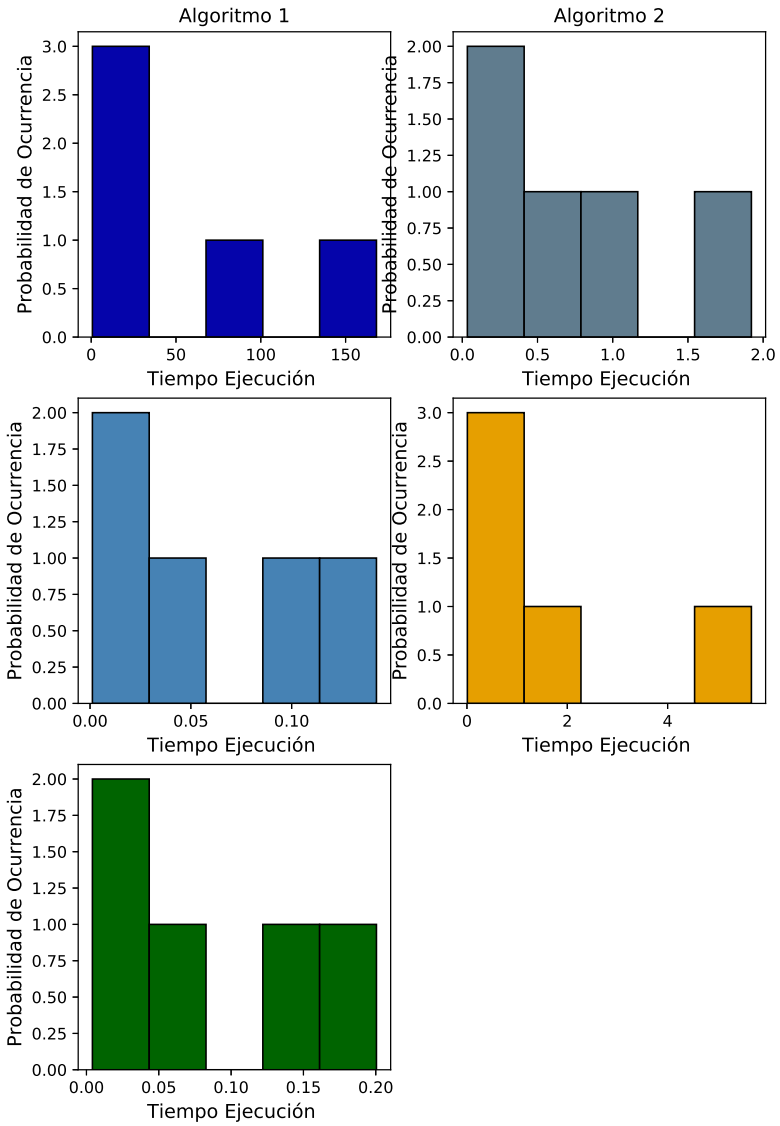

Figura 1: Histogramas

2. Scatter Plot 1

La figura 2 presenta el **scatter plot** que ilustra el comportamiento de los algoritmos en relación a la cantidad de arcos de los grafos. A continuación le sigue el código para obtener el gráfico del **scatter plot**. En la figura 2 el algoritmo **betweenness centrality** está representado por el símbolo + de color negro, el algoritmo **minimum spanning tree** está representado por un * de color verde, el algoritmo **greedy color** está representado por un rombo de color rojo, el algoritmo **max clique** está representado por círculos de color azul y el símbolo del algoritmo **maximal matching** corresponde al >de color amarillo.

2.1. Código

```
def guardarDatosenCSVParaObtenerHistograma(nombre):
    valores={'Alg1':[],
             'grafo':[],
             'media':[],
             'arcos':[],
             'nodos':[],
             }
    for j in range(5):
          valores["Alg1"].append("betweenness_centrality")
          valores["grafo"].append((nombre+str(j)))
          valores["media"].append((betweenness_centrality(nombre+str(j)+"
    .csv"))[0])
          valores["arcos"].append((betweenness_centrality(nombre+str(j)+"
    .csv"))[4])
          valores["nodos"].append((betweenness_centrality(nombre+str(j)+"
    .csv"))[3])
    for j in range(5):
        valores["Alg1"].append( "minimum_spanning_tree")
        valores["grafo"].append( (nombre+str(j)))
        valores["media"].append( (minimum_spanning_tree(nombre+str(j)+".
    csv"))[0])
        valores["arcos"].append( (minimum_spanning_tree(nombre+str(j)+".
    csv"))[4])
        valores["nodos"].append(minimum_spanning_tree(nombre+str(j)+".csv
    ")[3])
    for j in range(5):
        valores["Alg1"].append( "greedy_color")
        valores["grafo"].append( (nombre+str(j)))
        valores["media"].append( (greedy_color(nombre+str(j)+".csv"))[0])
        valores["arcos"].append( (greedy_color(nombre+str(j)+".csv"))[4])
        valores["nodos"].append(greedy_color(nombre+str(j)+".csv")[3])
    for j in range(5):
        valores["Alg1"].append( "max_clique")
        valores["grafo"].append( (nombre+str(j)))
        valores["media"].append( (max_clique(nombre+str(j)+".csv"))[0])
```

```
        valores["arcos"].append( (max_clique(nombre+str(j)+".csv"))[4])
        valores["nodos"].append(max_clique(nombre+str(j)+".csv")[3])
    for j in range(5):
        valores["Alg1"].append( "maximal_matching")
        valores["grafo"].append( (nombre+str(j)))
        valores["media"].append( (maximal_matching(nombre+str(j)+".csv"))[0])
        valores["arcos"].append( (maximal_matching(nombre+str(j)+".csv"))[4] )
        valores["nodos"].append(maximal_matching(nombre+str(j)+".csv")[3])
    df = pd.DataFrame(valores)
    df.to_csv("valores.csv", index=None)
```

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.mlab as m
import statistics as stats
data = pd.read_csv("VaLLoress.csv")
color_names = ["black", "green", "red", "blue", "yellow"]
#calculo = np.diff(data["media"]) / data["media"][:-1]
betweenness_centrality = data[(data["Alg1"] == "betweenness_centrality")]
minimun_spanning_tree= data[(data["Alg1"] == "minimum_spanning_tree")]
greedy_color = data[(data["Alg1"] == "greedy_color")]
max_clique = data[(data["Alg1"] == "max_clique")]
maximal_matching = data[(data["Alg1"] == "maximal_matching")]
size = (5 * data["arcos"][:5] / data["nodos"][:5])
figure, axis = plt.subplots(figsize=(10, 10))
axis.scatter(betweenness_centrality["media"], betweenness_centrality["arcos"],
            s=size, c=color_names, marker="+",
            label="Algoritmo Betweenness centrality", alpha=0.8, edgecolors='none')
axis.scatter(minimun_spanning_tree["media"], minimun_spanning_tree["arcos"],
            s=size, c=color_names, marker="*",
            label="Algoritmo Minimun Spanning Tree", alpha=0.8, edgecolors='none')
axis.scatter(greedy_color["media"], greedy_color["arcos"],
            s=size, c=color_names, marker="D",
            label="Algoritmo Greedy color", alpha=0.8, edgecolors='none')
axis.scatter(max_clique["media"], max_clique["arcos"],
            s=size, c=color_names, marker="o",
            label="Algoritmo Max Clique", alpha=0.8, edgecolors='none')
axis.scatter(maximal_matching ["media"], maximal_matching ["arcos"],
            s=size, c=color_names, marker=">",
            label="Algoritmo maximal_matching ", alpha=1, edgecolors='none')
```

```
axis.set_ylabel("Cantidad de arcos del grafo", fontsize=10)
axis.set_xlabel("Tiempo de ejecucion", fontsize=10)
plt.ylim((min(greedy_color["arcos"])-5, max(greedy_color["arcos"]) + 5))
plt.legend(loc="center right")
plt.axis('on')
plt.savefig("SPlOOTTTT.eps")
plt.show()
```

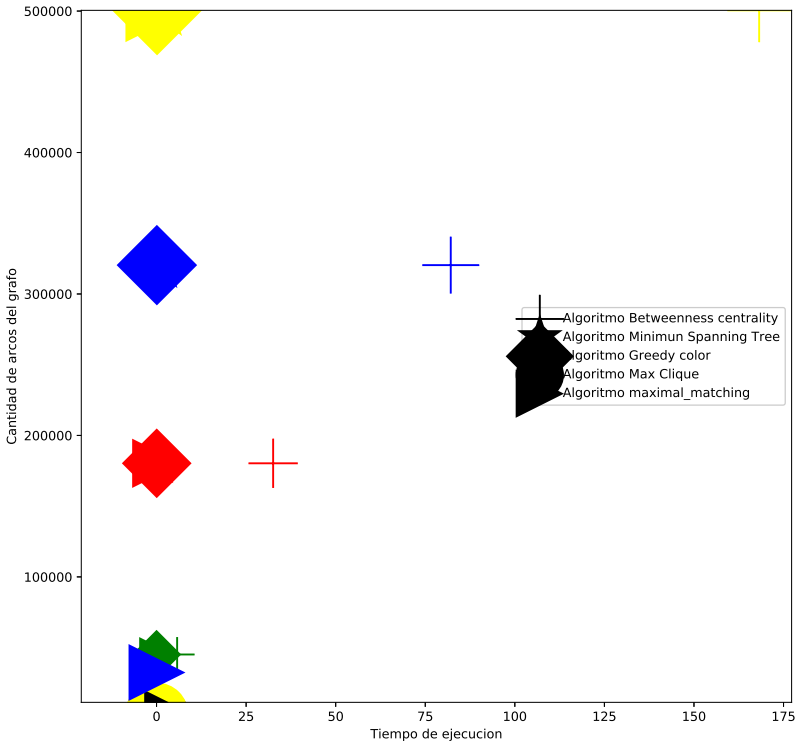

Figura 2: Scatter Plot 1

3. Scatter Plot 2

La figura 3 presenta el **scatter plot** que ilustra el comportamiento de los algoritmos en relación a la cantidad de nodos de los grafos, se aprecia que el algoritmo **betweenness centrality** presenta el mayor tiempo de ejecución

en relación con el resto de los algoritmos. Se obtuvieron tiempos de ejecución que oscilan entre 0.002 y 5 segundos como se refleja en la tabla del cuadro 1, aunque se obtuvieron valores superiores para el caso del algoritmo **betweenness centrality**. En todos los casos se ejecutaron primero 50 réplicas y luego otras 30 réplicas más, luego se hallaron la media, la mediana y la desviación estándar.

Se puede concluir que el comportamiento de los datos reflejados en la tabla y en las gráficas de Scatter plot es que a medida que aumenta la cantidad de nodos en los grafos aumenta el tiempo de ejecución de las réplicas.

3.1. Código

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.mlab as m
import statistics as stats
data = pd.read_csv("VaLLoress.csv")
color_names = ["black", "green", "red", "blue", "yellow"]
calculo = np.diff(data["media"]) / data["media"][:-1]
betweenness_centrality = data[(data["Alg1"] == "betweenness_centrality")]
minimun_spanning_tree= data[(data["Alg1"] == "minimum_spanning_tree")]
greedy_color = data[(data["Alg1"] == "greedy_color")]
max_clique = data[(data["Alg1"] == "max_clique")]
maximal_matching = data[(data["Alg1"] == "maximal_matching")]
size = (5 * data["arcos"][:5] / data["nodos"][:5])
figure, axis = plt.subplots(figsize=(8, 8))
axis.scatter(betweenness_centrality["media"], betweenness_centrality["
    nodos"],
            s=size, c=color_names, marker="+",
            label="Algoritmo Betweenness centrality", alpha=0.8,
    edgecolors='none')
axis.scatter(minimun_spanning_tree["media"], minimun_spanning_tree["nodos
    "],
            s=size, c=color_names, marker="*",
            label="Algoritmo Minimun Spanning Tree", alpha=0.8,
    edgecolors='none')
axis.scatter(greedy_color["media"], greedy_color["nodos"],
            s=size, c=color_names, marker="D",
            label="Algoritmo Greedy color", alpha=0.8, edgecolors='none')
axis.scatter(max_clique["media"], max_clique["nodos"],
            s=size, c=color_names, marker="o",
            label="Algoritmo Max Clique", alpha=0.8, edgecolors='none')
axis.scatter(maximal_matching ["media"], maximal_matching ["nodos"],
```

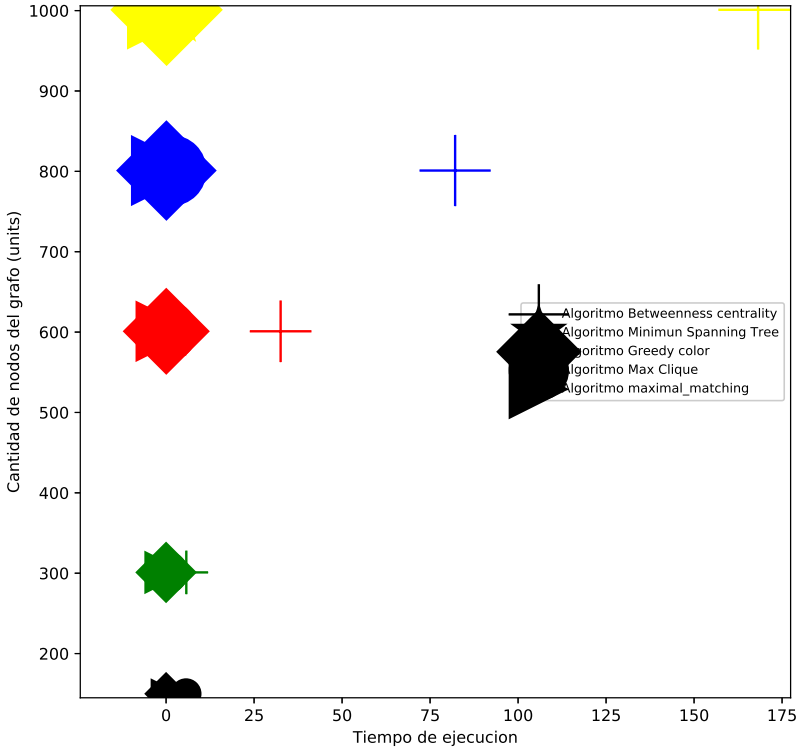

Figura 3: Scatter Plot 2

Referencias

[1] Matplotlib developers. `https://matplotlib.org`.

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

[3] Numpy developers. `https://www.numpy.org/`.

[4] Olivier Pomel. `https://pandas.pydata.org/`.

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