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 y para cada uno se corrieron los cinco algoritmos obteniendo 25 combinaciones. 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 5 histogramas, uno por cada algoritmo utilizado que son **betweenness centrality**, **minimum spanning tree**, **greedy color**, el algoritmo **max clique** y el algoritmo **maximal matching**.

El histograma del algoritmo 1 refleja que el comportamiento de los tiempos registrados no tiene una distribución nomal, o sea no tiene una distribución simétrica.

El histograma del algoritmo 2 refleja que el comportamiento dl tiempo registrado no sigue una distribución nomal, no tiene una distribución simétrica.

1.1. Código

```
1 import datetime
2 import matplotlib.pyplot as plt
3 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)
```

```
104     mediana=np.median(tiempos)
105     nodos=nx.number_of_nodes(b)
106     arcos=nx.number_of_edges(b)
107     salvar=[]
108     salvar.append(media)
109     salvar.append(desv)
110     salvar.append(mediana)
111     salvar.append(nodos)
112     salvar.append(arcos)
113     return salvar
114 
115 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':[],
```

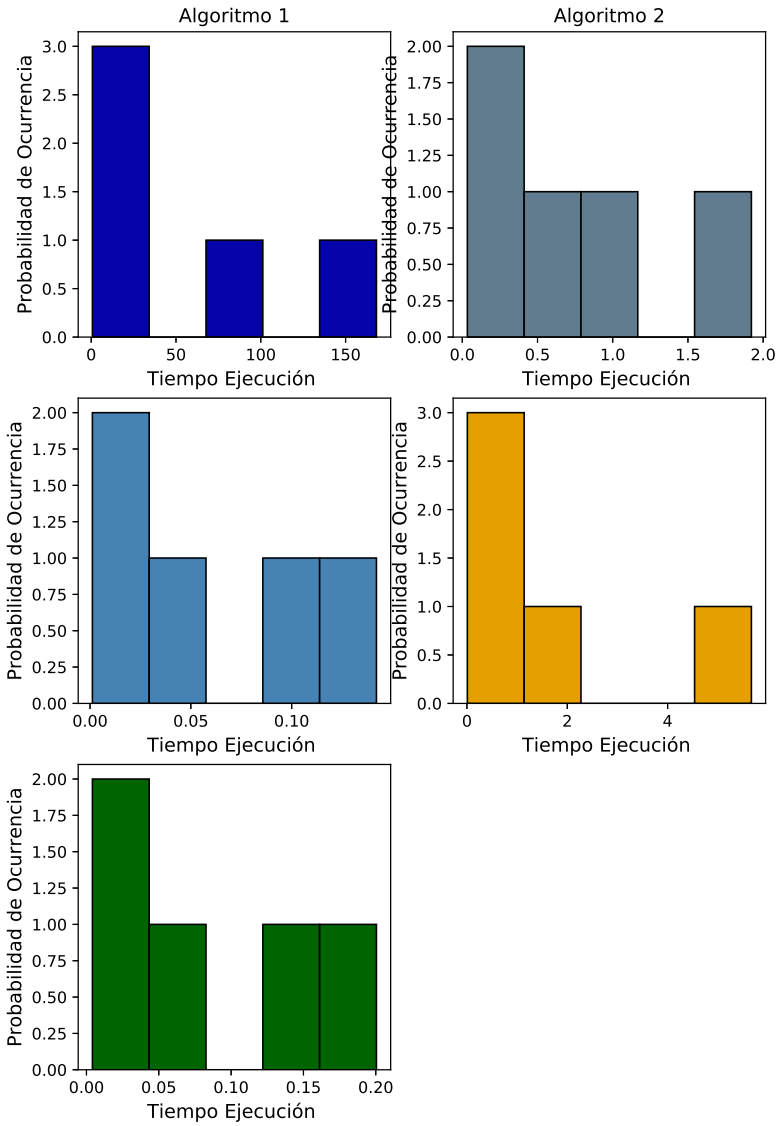

Figura 1: Histograma 1

2. Scatter Plot 1

La figura 2 presenta el **scatter plot** que ilustra el comportamiento de los algoritmos vs la cantidad de arcos de los grafos. A continuación le sigue el código para obtener el gráfico del **scatter plot**.

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="o",
            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')
```

```
36 plt.savefig("SPlOOTTTT.eps")
37 plt.show()
```

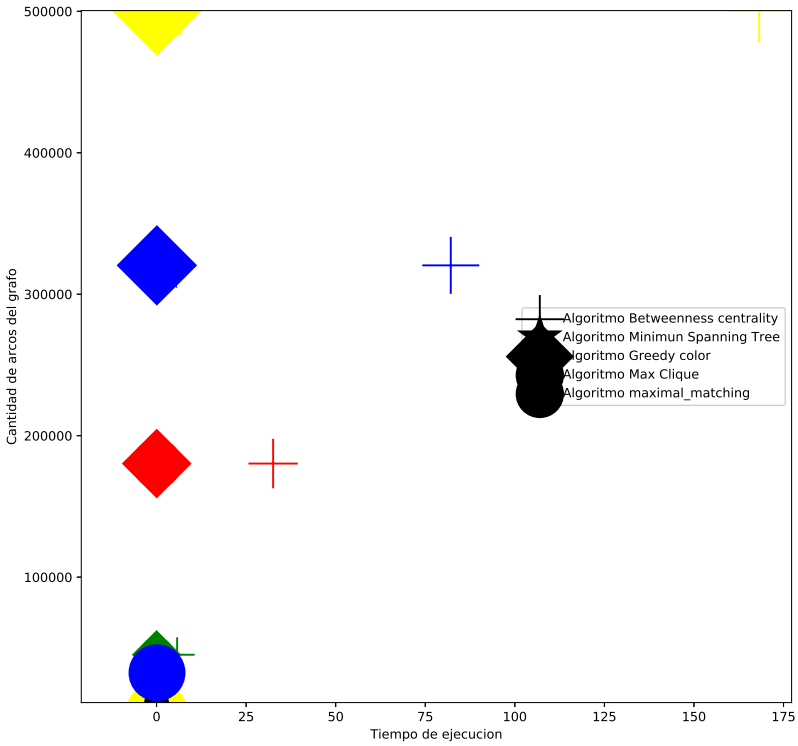

Figura 2: Scatter Plot 1

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 timepo de ejecución, para el resto de los algoritmos se obtuvieron tiempos de ejecución sobre los 3 segundos aproximadamente.

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"],
```

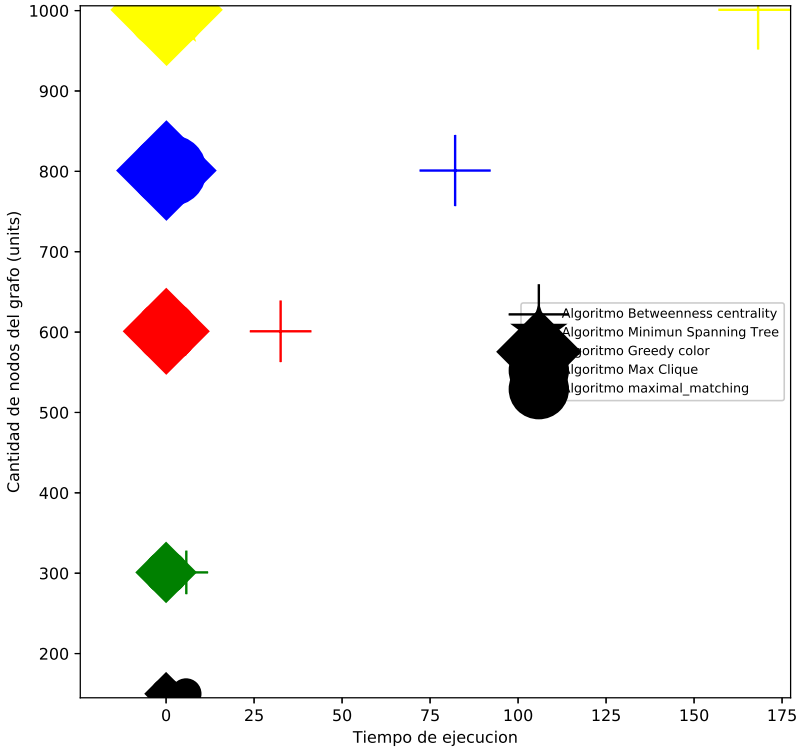

Figura 3: Scatter Plot 2

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`.