Tarea No.3: Rectificada

Dayli Machado (5275)

3 de junio de 2019

1. Grafos y algoritmos empleados

Para la realización de esta tarea se crearon nuevos grafos pues con los que se contaba de tareas anteriores no se podía realizar la medición del experimeinto y obtener valores medibles pues el número máximo de nodos con los que se contaba era de cinco en todos los grafos. A continuación se muestran las líneas de código que fueron empleadas para generar los grafos con los que se trabajó durante la medición de los algoritmos seleccionados.

```
def Generar(nombre, num_archivos, cant_nodos_min, probabilidad):
    G = nx.Graph()
    for r in range(num_archivos):
        print("Archivo", r)
        for i in range(cant_nodos_min):
           G.add_node(i)
            for j in range(i + 1, cant_nodos_min):
                crear_vertice = rnd.randint(0, 100) < probabilidad
                print(i, j, crear_vertice)
                if crear_vertice:
                    G.add_edge(i,j, distancia=rnd.randint(10,15))
            df = nx.to_pandas_adjacency(G, dtype=int, weight='distancia')
            df.to_csv(nombre+str(r)+".csv", index = None,header=None )
    return 0

#Generar("grafo", 5, 200, 25)

def Clicgenerar(nombre, num_archivos, cant_nodos_min, probabilidad):
    G = nx.Graph()
    for r in range(num_archivos):
        print("Archivo", r)
        for i in range(cant_nodos_min):
           G.add_node(i)
            for j in range(i + 1, cant_nodos_min):
                crear_vertice = rnd.randint(0, 100) < probabilidad
                print(i, j, crear_vertice)
                if crear_vertice:
                    G.add_edge(i,j, distancia=rnd.randint(10,15))
            df = nx.to_pandas_adjacency(G, dtype=int, weight='distancia')
            df.to_csv(nombre+str(r)+".csv", index = None,header=None )
    return 0

#Clicgenerar("clicgrafo", 5, 40, 10)

def Digenerar(nombre, num_archivos, cant_nodos_min, probabilidad):
    G = nx.DiGraph()
    for r in range(num_archivos):
        print("Archivo", r)
```

```
        for i in range(cant_nodos_min):
            G.add_node(i)
            for j in range(i + 1, cant_nodos_min):
                crear_vertice = rnd.randint(0, 100) < probabilidad
                print(i, j, crear_vertice)
                if crear_vertice:
                    G.add_edge(i,j, distancia=rnd.randint(10,15))
            df = nx.to_pandas_adjacency(G, dtype=int, weight='distancia')
            df.to_csv(nombre+str(r)+".csv", index = None,header=None )
    return 0

#Digenerar("digrafo", 5, 400, 25)
```

codigomadre.py

Se generaron cinco grafos: dirigidos y no dirigidos, ponderados, con la cantidad de nodos y arcos aleatorias, así como otro grupo de cinco grafos con una cantidad de nodos y arcos menor para algoritmos que así lo requerían.

Para la selección del tipo de algoritmo se tuvo en cuenta que combinaran con los grafos generados y permitieran obtener valores medibles de ejecución en cada uno. Fueron probados todos y de ellos los que mejores resultados de medición arrojaron fueron los siguientes:

- Centralidad de intermediación (en lo adelante BC según sus siglas en inglés *Betweenness centrality*)
- *Max clique* (en lo adelante MC)
- *Greedy* color (en lo adelante GC)
- Flujo Máximo (en lo adelante MF, por sus siglas en inglés *Maximum flow*)
- Árbol de expansión mínima (en lo adelante MS, por sus siglas en inglés Minimum spanning tree)

1.1. Breve descripción de los algoritmos medidos y su código

El algoritmo BC nos brinda la medida de centralidad de un grafo basado en la trayectoria más corta, este devuelve un diccionario de nodos con la medidad de centralidad [de NetworkX(a)]. La centralidad de la interrelación de un nodo es la suma de la fracción de las rutas más cortas de todos los pares que pasan por él [de NetworkX(b)]. Puede usarse en grafos dirigidos y no dirigidos.

El algoritmo MC devuelve el subgrafo más grande que encuentre, se recomienda su uso en grafos no dirigidos, es uno de los algoritmos que más tiempo demora en su ejecución.

Por su parte el algoritmo GC, lo que hace es colorear un nodo usando diferentes estrategias de coloración que se le pasan como un parámetro dentro de una función. Devuelve un diccionario con claves que representan nodos y valores que representan la coloración correspondiente [NetworkX(a)]. Puede emplearse en grafos dirigidos y no dirigidos.

El algoritmo MF 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 [NetworkX(b)]. Se recomienda emplearlo en grafos dirigidos.

El algoritmo MS tree devuelve la conexión entre los nodos de modo que la unión entre ellos es un árbol no dirigido ponderado cuya suma de pesos de cada arco es la menor posible [NetworkX(c)].

A continuación se muestra el fragmento de código desarrollado para la medición del tiempo por cada algoritmo:

```
def betweenness_centrality(name):
    dr = pd.read_csv(name,header=None)
    F = nx.from_pandas_adjacency(dr, create_using= nx.Graph())
    tiempo=[]
    for i in range(30):
        tiempo_inicial = dt.datetime.now()
        d= nx.betweenness_centrality(F) #  aca estoy guardando el resultado del algoritmo y como no lo uso paranada me da
        tiempo_final = dt.datetime.now()
        tiempo_ejecucion = (tiempo_final - tiempo_inicial).total_seconds()
        tiempo.append(tiempo_ejecucion)

    media=nup.mean(tiempo)
    desv=nup.std(tiempo)
    mediana=nup.median(tiempo)
    salvar=[]
    salvar.append(media)
    salvar.append(desv)
    salvar.append(mediana)
    df = pd.DataFrame(salvar)
    df.to_csv("bet_"+name, index = None,header=None )
    print("terminado betweeness")

def max_clique(name):
    dr = pd.read_csv(name,header=None)
    F = nx.from_pandas_adjacency(dr, create_using= nx.Graph())
    tiempo=[]
    for i in range(30):
        tiempo_inicial = dt.datetime.now()
        d= nx.make_max_clique_graph(F,create_using=None) #  aca estoy guardando el resultado del algoritmo y como no lo uso paranada me da
        tiempo_final = dt.datetime.now()
        tiempo_ejecucion = (tiempo_final - tiempo_inicial).total_seconds()
        tiempo.append(tiempo_ejecucion)

    media=nup.mean(tiempo)
    desv=nup.std(tiempo)
    mediana=nup.median(tiempo)
    salvar=[]
    salvar.append(media)
    salvar.append(desv)
    salvar.append(mediana)
    df = pd.DataFrame(salvar)
    df.to_csv("clique_"+name, index = None,header=None )
    print("terminado maxclique")


def greedy_color(name):
    dr = pd.read_csv(name,header=None)
    F = nx.from_pandas_adjacency(dr, create_using= nx.Graph())
    tiempo=[]
    for i in range(300):
        tiempo_inicial = dt.datetime.now()
        d= nx.greedy_color(F)
        tiempo_final = dt.datetime.now()
        tiempo_ejecucion = (tiempo_final - tiempo_inicial).total_seconds()
        tiempo.append(tiempo_ejecucion)

    media=nup.mean(tiempo)
    desv=nup.std(tiempo)
    mediana=nup.median(tiempo)
    salvar=[]
```

```
    salvar.append(media)
    salvar.append(desv)
    salvar.append(mediana)
    df = pd.DataFrame(salvar)
    df.to_csv("greed_"+name, index = None,header=None )
print("Terminado greedy")

def maximum_flow (name):
    dr = pd.read_csv(name,header=None)
    F = nx.from_pandas_adjacency(dr, create_using= nx.DiGraph())
    tiempo=[]
    for i in range(30):
        tiempo_inicial = dt.datetime.now()
        d= nx.maximum_flow (F, 4, 2) #  aca estoy guardando el resultado del algoritmo
     y como no lo uso paranada me da
        tiempo_final = dt.datetime.now()
        tiempo_ejecucion = (tiempo_final - tiempo_inicial).total_seconds()
        tiempo.append(tiempo_ejecucion)

    media=nup.mean(tiempo)
    desv=nup.std(tiempo)
    mediana=nup.median(tiempo)
    salvar=[]
    salvar.append(media)
    salvar.append(desv)
    salvar.append(mediana)
    df = pd.DataFrame(salvar)
    df.to_csv("maxflow_"+name, index = None,header=None )
print ("terminado maximun flow")

def minimum_spanning_tree(name):
    dr = pd.read_csv(name,header=None)
    F = nx.from_pandas_adjacency(dr, create_using= nx.Graph())
    tiempo=[]
    for i in range(30):
        tiempo_inicial = dt.datetime.now()
        d= nx.minimum_spanning_tree(F) #  aca estoy guardando el resultado del
    algoritmo y como no lo uso paranada me da
        tiempo_final = dt.datetime.now()
        tiempo_ejecucion = (tiempo_final - tiempo_inicial).total_seconds()
        tiempo.append(tiempo_ejecucion)

    media=nup.mean(tiempo)
    desv=nup.std(tiempo)
    mediana=nup.median(tiempo)
    salvar=[]
    salvar.append(media)
    salvar.append(desv)
    salvar.append(mediana)
    df = pd.DataFrame(salvar)
    df.to_csv("spanning_tree_"+name, index = None,header=None )
print ("terminado spanning_tree ")
```

codigomadre.py

2. Resultados

Se realizó la medición de los tiempos de los algoritmos teniendo en cuenta la media y la desviación estándar en cada caso.

Para obtener la cantidad de nodos y arístas de cada grafo empleado se desarrolló el siguiente código:

```
def GrafosElementos(cant):
    grafos={
              "clicgrafoN":[],
              "digrafoN":[],
              "grafoN":[],
              "clicgrafoE":[],
              "digrafoE":[],
              "grafoE":[],
            }
    for i in range(cant):
        dr = pd.read_csv("clicgrafo"+str(i)+".csv",header=None)
        F = nx.from_pandas_adjacency(dr, create_using= nx.Graph())
        grafos[ "clicgrafoN"].append(F.number_of_nodes())

        dr = pd.read_csv("digrafo"+str(i)+".csv",header=None)
        F = nx.from_pandas_adjacency(dr, create_using= nx.Graph())
        grafos[ "digrafoN"].append(F.number_of_nodes())

        dr = pd.read_csv("grafo"+str(i)+".csv",header=None)
        F = nx.from_pandas_adjacency(dr, create_using= nx.Graph())
        grafos[ "grafoN"].append(F.number_of_nodes())

        dr = pd.read_csv("clicgrafo"+str(i)+".csv",header=None)
        F = nx.from_pandas_adjacency(dr, create_using= nx.Graph())
        grafos[ "clicgrafoE"].append(F.number_of_edges())

        dr = pd.read_csv("digrafo"+str(i)+".csv",header=None)
        F = nx.from_pandas_adjacency(dr, create_using= nx.Graph())
        grafos[ "digrafoE"].append(F.number_of_edges())

        dr = pd.read_csv("grafo"+str(i)+".csv",header=None)
        F = nx.from_pandas_adjacency(dr, create_using= nx.Graph())
        grafos[ "grafoE"].append(F.number_of_edges())

    df = pd.DataFrame(grafos)
    df.to_csv("cantNE.csv" )

#GrafosElementos(5)
```

codigomadre.py

Para leer de los archivos .csv los valores de media y desviación estándar se usó el siguiente código:

```
def lee_valores(cant):

    algorit={

              "Media_A1bc":[],
              "Media_A2mc":[],
              "Media_A3gc":[],
              "Media_A4mf":[],
              "Media_A5mst":[],
              "Desvi_A1bc":[],
              "Desvi_A2mc":[],
              "Desvi_A3gc":[],
              "Desvi_A4mf":[],
```

```
            "Desvi_A5mst":[]
              }

    for i in range(cant):

        dr = pd.read_csv("bet_grafo"+str(i)+".csv",header=None,)
#         print(dr[0][0])
        algorit[ "Media_A1bc"].append(dr[0][0])

        dr = pd.read_csv("clique_clicgrafo"+str(i)+".csv",header=None)

        algorit["Media_A2mc"].append(dr[0][0])

        dr = pd.read_csv("greed_digrafo"+str(i)+".csv",header=None)
        algorit["Media_A3gc"].append(dr[0][0])

        dr = pd.read_csv("maxflow_digrafo"+str(i)+".csv",header=None)
        algorit["Media_A4mf"].append(dr[0][0])

        dr = pd.read_csv("spanning_tree_grafo"+str(i)+".csv",header=None)
        algorit["Media_A5mst"].append(dr[0][0])

        dr = pd.read_csv("bet_grafo"+str(i)+".csv",header=None)
#         print(dr[0][1])
        algorit[ "Desvi_A1bc"].append(dr[0][1])

        dr = pd.read_csv("clique_clicgrafo"+str(i)+".csv",header=None)
        algorit["Desvi_A2mc"].append(dr[0][1])

        dr = pd.read_csv("greed_digrafo"+str(i)+".csv",header=None)
        algorit["Desvi_A3gc"].append(dr[0][1])

        dr = pd.read_csv("maxflow_digrafo"+str(i)+".csv",header=None)
        algorit["Desvi_A4mf"].append(dr[0][1])

        dr = pd.read_csv("spanning_tree_grafo"+str(i)+".csv",header=None)
        algorit["Desvi_A5mst"].append(dr[0][1])

    df = pd.DataFrame(algorit)
    df.to_csv("algori_med.csv" )


lee_valores(5)
```

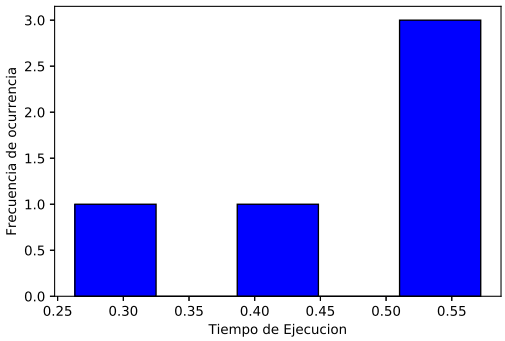
codigomadre.py

Luego se realizaron los histogramas que reflejan la variación del valor de la media por algoritmo empleado. Seguidamente se muestra un fragmento del códico y los histogramas por algoritmos se muestran en la figura 1 de la página 8. En la gráfica se observa que en la mayoría de los algoritmos existe tendencia a ubicarse hacia los valores extremos, excepto en el algoritmo GC, que se aprecia una cantidad de valores igual en cada barra.

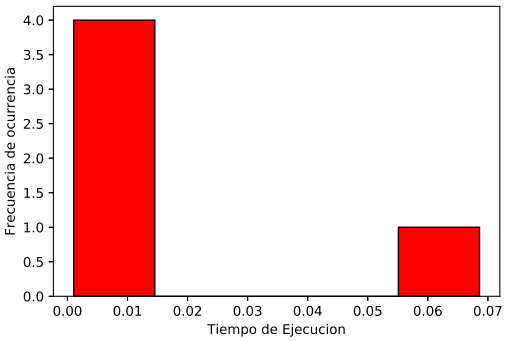
```
import random as rnd
import networkx as nx
import matplotlib.pyplot as plt
import numpy as nup
import datetime as dt
import pandas as pd
import statistics as stats

data = pd.read_csv("1.csv")



```

```
plt.hist(nup.log1p(data["Media_A1bc"]), 5, facecolor='blue', alpha=0.5, edgecolor = 'black',  linewidth=1)
plt.ylabel('Frecuencia de ocurrencia')
plt.xlabel('Tiempo de Ejecucion')
plt.savefig("Imagenes/Histograma1.eps",bbox_inches='tight')
plt.show()


plt.hist(nup.log1p(data["Media_A2mc"]), 5, facecolor='red', alpha=0.5,  edgecolor = 'black',  linewidth=1)
plt.ylabel('Frecuencia de ocurrencia')
plt.xlabel('Tiempo de Ejecucion')
plt.savefig("Imagenes/Histograma2.eps",bbox_inches='tight')
plt.show()

plt.hist(nup.log1p(data["Media_A3gc"]), 5, facecolor='green', alpha=0.5, edgecolor = 'black',  linewidth=1)
plt.ylabel('Frecuencia de ocurrencia')
plt.xlabel('Tiempo de Ejecucion')
plt.savefig("Imagenes/Histograma3.eps",bbox_inches='tight')
plt.show()
```

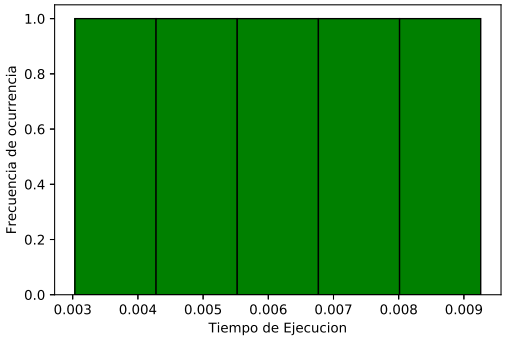
Histogramas.py



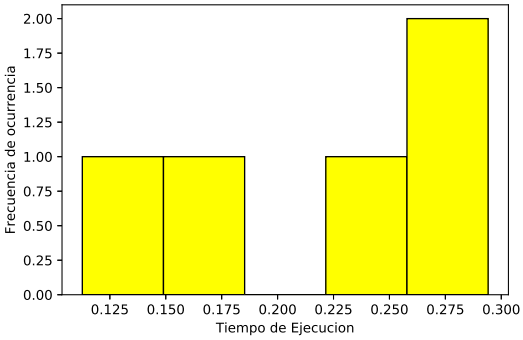
(a) *BC*



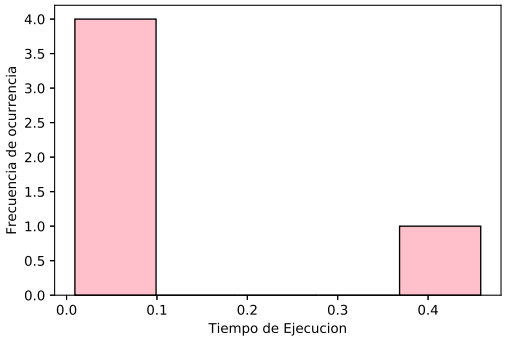
(b) *MC*



(c) *GC*



(d) *MF*



(e) *MS*

Figura 1: Histograma de cada uno de los cinco algoritmo con los cinco grafos generados.

Para concluir se graficaron en un diagrama de dispersión el tiempo medio que se demora cada algoritmo en función de la cantidad de nodos y aristas respectivamente.

En la figura 2 de la página 10, se muestra la relación entre la media de cada algoritmo y la cantidad de nodos que se emplearon por grafo. De esta se puede apreciar que de los algoritmos que se corrieron con 200 nodos que fueron el MS y el BC, el que presenta valores más elevados de la media en cada una de las veces que se corrió el algoritmo fue el BC. De los que se corrieron para 400 nodos que fueron los algoritmos GC y MF, el que posee mayores valores de la media fue el MF. Finalmente se corrió con una cantidad menor el algoritmo MC, pues es el que más tiempo consumía en la medición, arrojando lógicamente menores valores en la media, sin embargo a pesar de tener solo 50 nodos al compararlo con el algoritmo MS que se corrió con 400 nodos, los valores en la media que arroja son similares, lo que demuestra que el MC es mucho más lento en su tiempo de ejecución que el MS. De lo anterior se destaca como oportunidad de mejora homogenizar la cantidad de nodos que se empleen en los grafos para tener una comparación más cercana a la realidad.

A continuación se muestra el código donde aparece identificado la figura y color a cúal algoritmo corresponde.

```
import matplotlib.patches as mpatches

data = pd.read_csv("1.csv")

figure, axes = plt.subplots(figsize=(10, 10))
x=data["Media_A1bc"]
y=data["grafoN"]
xerr=data["Desvi_A1bc"]

axes.errorbar(x, y,  xerr=xerr, fmt='D',color="#932525", alpha=1, label="BC")
print(y)

x=data["Media_A2mc"]
y=data["clicgrafoN"]
xerr=data["Desvi_A2mc"]

axes.errorbar(x, y, xerr=xerr, fmt='s',color="#129f10", alpha=1, label="MC")
print(y)

x=data["Media_A3gc"]
y=data["digrafoN"]
xerr=data["Desvi_A3gc"]

axes.errorbar(x, y, xerr=xerr,fmt='8',color="#093ea8", alpha=1, label="GC")
print(y)

x=data["Media_A4mf"]
y=data["digrafoN"]
xerr=data["Desvi_A4mf"]

axes.errorbar(x, y,xerr=xerr, fmt='>',color="#adcb18", alpha=1, label="MF")
print(y)

x=data["Media_A5mst"]
y=data["grafoN"]
xerr=data["Desvi_A5mst"]

axes.errorbar(x, y,xerr=xerr, fmt='o',color="#ee9110", alpha=1, label="MST")
print(y)
```

Scatterplot.py

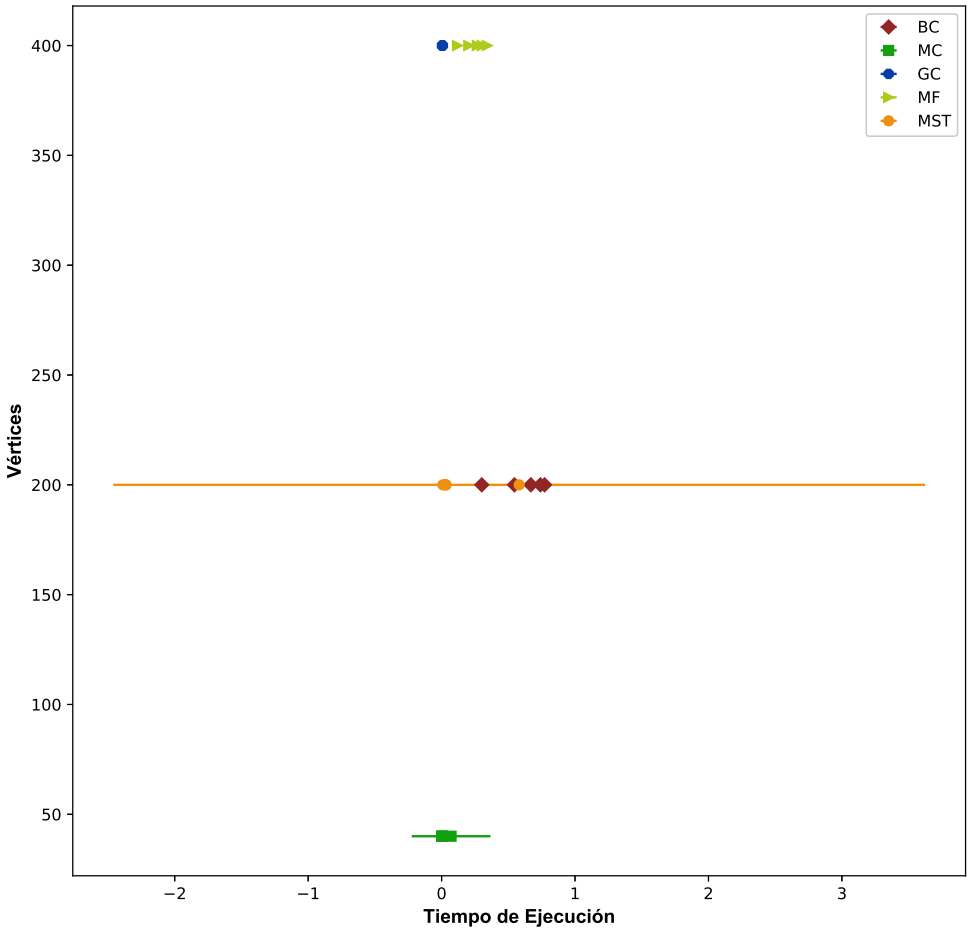


Figura 2: Diagrama de disperción (cant. nodos vs media-algoritmo)

En la figura 3 de la página 12 se muestra la relación entre la media de cada algoritmo y la cantidad de arcos que se emplearon por grafo. En esta se aprecia que como tendencia general en los algoritmos medidos, al aumentar el número de arcos aumenta la media, el que más aumenta en su media es el algoritmo BC, en el algoritmo GC el valor de la media permanece casi constante a pesar del aumento de arcos, lo cual tiene sentido por el modo en que opera el mismo, el algoritmo MS actua de manera similar. Para el MC es casi despreciable la cantidad de arcos, debido a la demora del mismo se decidió disminuirlos.

Seguidamente se muestra el fragmento de código:

```
1 axes.legend()
2 plt.savefig("Imagenes/Fig2.eps")
3 plt.show()
4
```

```
5
6
7  figure, axes = plt.subplots(figsize=(10, 10))
8
9  x=data["Media_A1bc"]
10 y=data["grafoE"]
11 xerr=data["Desvi_A1bc"]
12 axes.errorbar(x, y, xerr=xerr, fmt='D',color="#932525", alpha=1, label="BC")
13 print(y)
14
15 x=data["Media_A2mc"]
16 y=data["clicgrafoE"]
17 xerr=data["Desvi_A2mc"]
18 axes.errorbar(x, y, xerr=xerr, fmt='s',color="#129f10", alpha=1, label="MC")
19 print(y)
20
21 x=data["Media_A3gc"]
22 y=data["digrafoE"]
23 xerr=data["Desvi_A3gc"]
24 axes.errorbar(x, y, xerr=xerr,fmt='8',color="#093ea8", alpha=1, label="GC")
25 print(y)
26
27 x=data["Media_A4mf"]
28 y=data["digrafoE"]
29 xerr=data["Desvi_A4mf"]
30 axes.errorbar(x, y, xerr=xerr, fmt='>',color="#adcb18", alpha=1, label="MF")
31 print(y)
32
33 x=data["Media_A5mst"]
34 y=data["grafoE"]
35 xerr=data["Desvi_A5mst"]
36 axes.errorbar(x, y,fmt='o',color="#ee9110", alpha=1, label="MST")
37 print(y)
38
39 axes.set_ylabel("Aritas", fontsize=12, fontfamily="arial", fontweight="bold")
40 axes.set_xlabel("Tiempo de Ejecucion", fontsize=12, fontfamily="arial", fontweight="
       bold")
```

Scatterplot.py

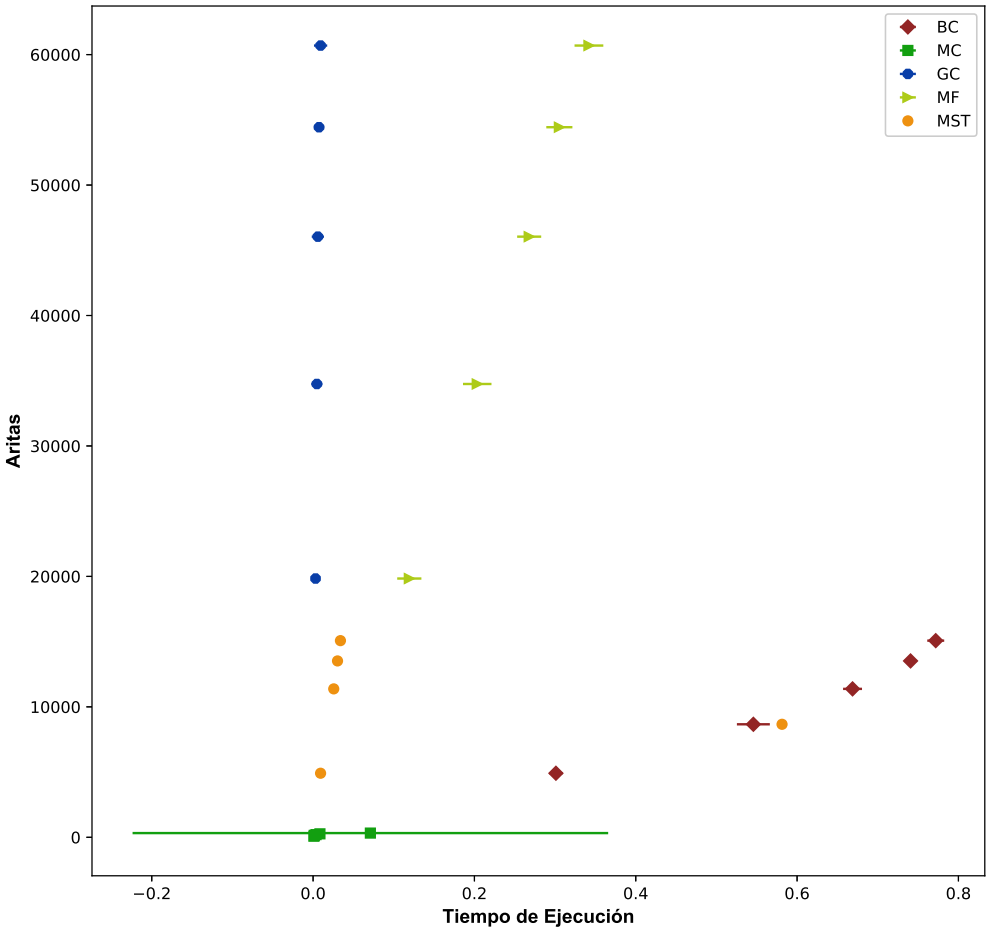


Figura 3: Diagrama de disperción (cant. arcos vs media-algoritmo)

Referencias

[de NetworkX(a)] Desarrolladores de NetworkX. https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.centrality.betweenness_centrality.html, a. Accessed: 2019-03-10.

[de NetworkX(b)] Desarrolladores de NetworkX. https://networkx.github.io/documentation/networkx-1.10/_modules/networkx/algorithms/centrality/betweenness.html.betweenness_centrality, b. Accessed: 2019-03-18.

[NetworkX(a)] Desarrolladores NetworkX. https://networkx.github.io/documentation/networkx-1.10/_modules/networkx/algorithms/coloring/greedy_coloring.html#greedy_color, a. Accessed: 2019-03-17.

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

[NetworkX(c)] Desarrolladores NetworkX. https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.mst.minimum_spanning_tree.html, c. Accessed: 2019-03-17.