Tarea 3

5280

3 de junio de 2019

Introducción

En este trabajo se realizan mediciones sobre los algoritmos para grafos de NetworkX [2]:

- `betweenness_centrality`: Ofrece una medida de centralidad de un grafo, la cual devuelve como un diccionario de nodos, con la medidad de centralidad. Puede usarse en grafos dirigidos y no dirigidos.
- `maximal_matching`: Devuelve un conjunto de nodos máximo conjunto posible de aristas independientes, que no tienen nodos en común.
- `greedy_color`: Colorea los nodo usando diferentes estrategias. Devuelve un diccionario con claves que representan nodos y valores que representan la coloración. Puede emplearse en grafos dirigidos y no dirigidos.
- `make_max_clique_graph`: Devuelve el subgrafo más grande que encuentra. Debe usarse en grafos no dirigidos.
- `strongly_connected_components`: Se utiliza para grafos dirigidos. Devuelve un generador de conjunto de nodos, uno por cada componente fuertemente conectado.

A los cuatro primeros algoritmos se les puede pasar como parámetro grafos no dirigidos ponderados, lo que resulta conveniente pues permite emplear el mismo conjuntos de grafos para los cuatro algoritmos. El algoritmo `strongly_connected_components` requiere grafos dirigidos, por lo que para este se emplea otro conjunto de grafos.

Generación de grafos

Todos los grafos son generados aleatoriamente empleando las funciones `GenerateGraph` y `GenerateDiGraph`. A estas funciones se le pasan los siguientes parámetros:

- `nameToSave`: Prefijo de la dirección con la que se quieren guardar los grafos. El código añadirá números consecutivos a este prefijo para cada grafo, comenzando por 0.
- `Smin`: Número mínimo de nodos que tendrán los gráfos a generar.
- `Smax`: Número máximo de nodos que tendrán los gráfos a generar.

- `max_weight`: Peso máximo que tendrán las aristas.
- `numberOfGraphs`: Cantidad de grafos a generar.

Los grafos son convertidos a `DataFrame` de la librería pandas [3] y guardados en formato `.csv`.

```
def GenerateGraph(nameToSave,Smin,Smax,max_weight,numberOfGraphs):
    for h in range(numberOfGraphs):
        G=nx.Graph()
        size=rdm.randint(Smin,Smax)
        for i in range(size):
            for j in range(i, size):
                if rdm.randint(0,int(size/20))==0:
                    G.add_edge(i,j, weight=rdm.randint(1,max_weight))
        df = pd.DataFrame()
        df = nx.to_pandas_adjacency(G, dtype=int, weight='weight')
        df.to_csv(nameToSave+str(h)+".csv")

def GenerateDiGraph(nameToSave,Smin,Smax,max_weight,numberOfGraphs):
    for h in range(numberOfGraphs):
        G=nx.DiGraph()
        size=rdm.randint(Smin,Smax)
        for i in range(size):
            for j in range(i, size):
                if rdm.randint(0,int(size/50))==0:
                    G.add_edge(i,j, weight=rdm.randint(1,max_weight))
                elif rdm.randint(0,int(size/30))==1:
                    G.add_edge(j,i, weight=rdm.randint(1,max_weight))
        df = pd.DataFrame()
        df = nx.to_pandas_adjacency(G, dtype=int, weight='weight')
        df.to_csv(nameToSave+str(h)+".csv")
```

Tarea3.py

Medición del tiempo de ejecución

Para cada uno de los algoritmos seleccionados se realiza una función que recibe un grafo como parámetro, ejecuta el algoritmo y devuelve el tiempo de ejecución del mismo.

```
def BetCen(graph):
    start_time=time()
    R = nx.betweenness_centrality(graph, weight= 'size', normalized=False)
    time_elapsed = time() - start_time
    return time_elapsed

def MinMaxMat(graph):
    start_time=time()
    for i in range (100):
        R = nx.maximal_matching(graph)
    time_elapsed = time() - start_time
    return time_elapsed

def GreedyColor(graph):
    start_time=time()
    for i in range (160):
        R = nx.greedy_color(graph, strategy='largest_first', interchange=False)
    time_elapsed = time() - start_time
    return time_elapsed

def MaxClique(graph):
    start_time=time()
```

```
    R = nx.make_max_clique_graph(graph, create_using=None)
    time_elapsed = time() - start_time
    return time_elapsed

def StronglyC(graph):
    start_time=time()
    for i in range(99000):
        R = nx.strongly_connected_components(graph)
    time_elapsed = time() - start_time
    return time_elapsed
```

Tarea3.py

Estas funciones son ejecutadas mediante `RunAll`, función a la que se le pasan los siguientes parámetros:

- `runs`: Número de veces que se quieren ejecutar los algoritmos.
- `numAlgorithms`: Cantidad de algoritmos a emplear.
- *numGraphs*: Cantidad de grafos que se va a correr para cada algoritmo.
- `name`: Formato del nombre con el que fueron guardados los grafos no dirigidos.
- `nameDi`: Formato de nombre con el que fueron guardados los grafos dirigidos.
- `matrix`: Este parámetro es un diccionario vacío cuyos índices son los números del 0 al 24.

```
def RunAll(runs,numAlgorithms,numGraphs,name,nameDi,matrix):
    combinations=[]
    for i in range(numAlgorithms-1):
        for j in range (numGraphs):
            combinations.append([i,name+str(j)+".csv"])
    for j in range (numGraphs):
        combinations.append([numAlgorithms-1,nameDi+str(j)+".csv"])
    for j in range(runs):
        np.random.shuffle(combinations)
        for i in combinations:
            if i[0]==0:
                for n in range(numGraphs):
                    if i[1]==name+str(n)+".csv":
                        matrix[str(n)].append(BetCen(ReadGraph(i[1])))
            if i[0]==1:
                for n in range(numGraphs):
                    if i[1]== name+str(n)+".csv":
                        matrix[str(n+5)].append(MinMaxMat(ReadGraph(i[1])))
            if i[0]==2:
                for n in range(numGraphs):
                    if i[1]== name+str(n)+".csv":
                        matrix[str(n+10)].append(GreedyColor(ReadGraph(i[1])))
            if i[0]==3:
                for n in range(numGraphs):
                    if i[1]== name+str(n)+".csv":
                        matrix[str(n+15)].append(MaxClique(ReadGraph(i[1])))
            if i[0]==4:
                for n in range(numGraphs):
                    if i[1]== "directed"+str(n)+".csv":
                        matrix[str(n+20)].append(StronglyC(ReadDiGraph(i[1])))
    df = pd.DataFrame(matrix)
    df.to_csv("matrix.csv")
```

Tarea3.py

Este código crea una lista con las 25 combinaciones de algoritmo-grafo y para cada repetición cambia la ubicación de los elementos de la lista para garantizar la aleatoricidad del proceso. Una vez terminadas todas las mediciones, son guardadas en el archivo `Matrix.csv`

Cálculo de parámetros

Con la función `MediaDesv` se calcula la media y la desviación estándar para cada algoritmo. Esta función extrae además el número de nodos y aristas de los grafos creados, información necesaria para la creación de las gráficas de dispersión.

```
def MediaDesv (adress,runs,numberOfGraphs,numAlgorithms,name,nameDi):
    media = {
             'Media0': [],
             'Media1': [],
             'Media2': [],
             'Media3': [],
             'Media4': [],
              }
    standar={'Standar0': [],
             'Standar1': [],
             'Standar2': [],
             'Standar3': [],
             'Standar4': [],
             }
    grafos={
             'EdgesUndirected':[],
             'NodesUndirected':[],
             'EdgesDirected':[],
             'NodesDirected':[]
             }
    matrix = pd.read_csv(adress)
    for n in range(5):
        grafos['EdgesUndirected'].append(ReadGraph (name+str(n)+".csv").
    number_of_edges())
        grafos['NodesUndirected'].append(ReadGraph (name+str(n)+".csv").
    number_of_nodes())
        grafos['EdgesDirected'].append(ReadGraph (nameDi+str(n)+".csv").
    number_of_edges())
        grafos['NodesDirected'].append(ReadGraph (nameDi+str(n)+".csv").
    number_of_nodes())
        for i in range(5):
            media['Media'+str(i)].append(np.mean(matrix[str(n+(5*i))]))
            standar['Standar'+str(i)].append(np.std(matrix[str(n+(5*i))]))
    df=pd.DataFrame(media)
    df.to_csv("Media.csv", index=None)
    df=pd.DataFrame(standar,)
    df.to_csv("Standar.csv", index=None)
    df=pd.DataFrame(grafos)
    df.to_csv("Grafos.csv", index=None)
```

Tarea3.py

Estos parámetros son guardados en formato `.csv` para su posterior utilización.

Histograma

Una vez obtenidos los valores promedios de tiempo de ejecución para cada combinación algoritmo-grafo, se grafica un histograma con los resultados, empleando la librería Matplotlib[1]. En la figura 1 se observan los histogramas correspondientes a cada algoritmo.

Gráficas de dispersión

En la figura 2 se observa la gráfica de dispersión en la que el eje horizontal corresponde al tiempo promedio de ejecución y el eje vertical al el número de nodos del grafo. En la gráfica de dispersión de la figura 3, el eje horizontal corresponde de igual manera al tiempo de ejecución de los algoritmos, mientras que el vertical corresponde al número de aristas de los grafos.

Leyenda de formas:

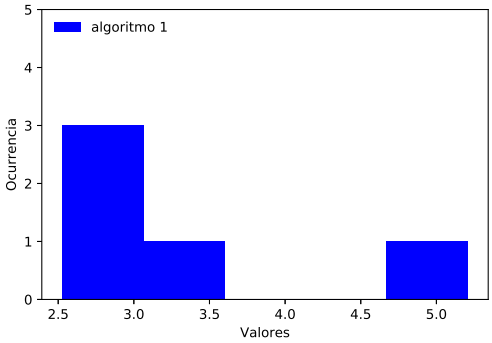
- `betweenness_centrality`=rombo
- `maximal_matching`=círculo
- `greedy_color`=estrella
- `make_max_clique_graph`=triángulo
- `strongly_connected_components`=cruz

Leyenda de colores:

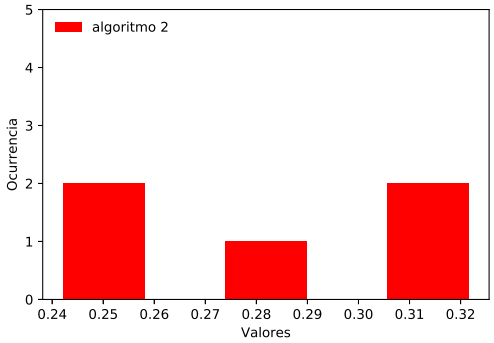
- `Grafo0`=amarillo
- `Grafo1`=rojo
- `Grafo2`=azul
- `Grafo3`=negro
- `Grafo4`=verde

Conclusiones

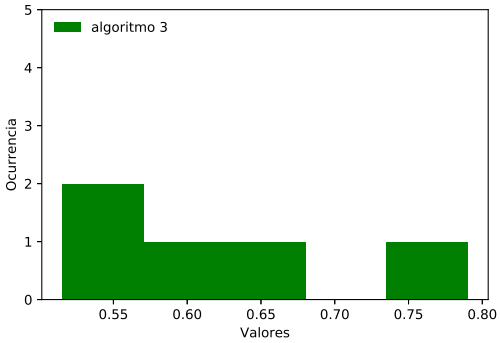
De manera general, pudo comprobarse que a medida que aumenta el número de nodos y/o aristas, aumenta el tiempo de ejecución de los algoritmos.



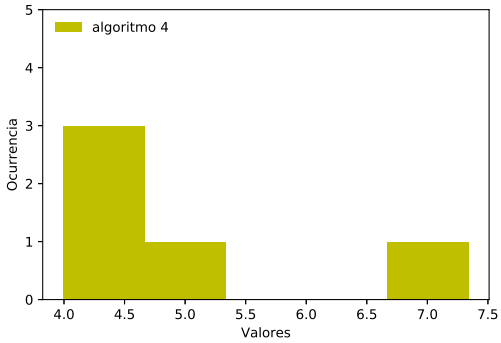
(a) `betweenness_centrality`



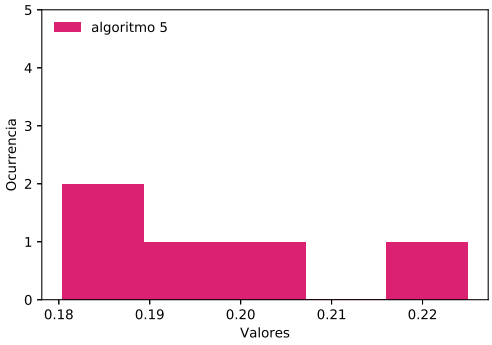
(b) `maximal_matching`



(c) `greedy_color`



(d) `strongly_connected_components`



(e) `Dfs tree`

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

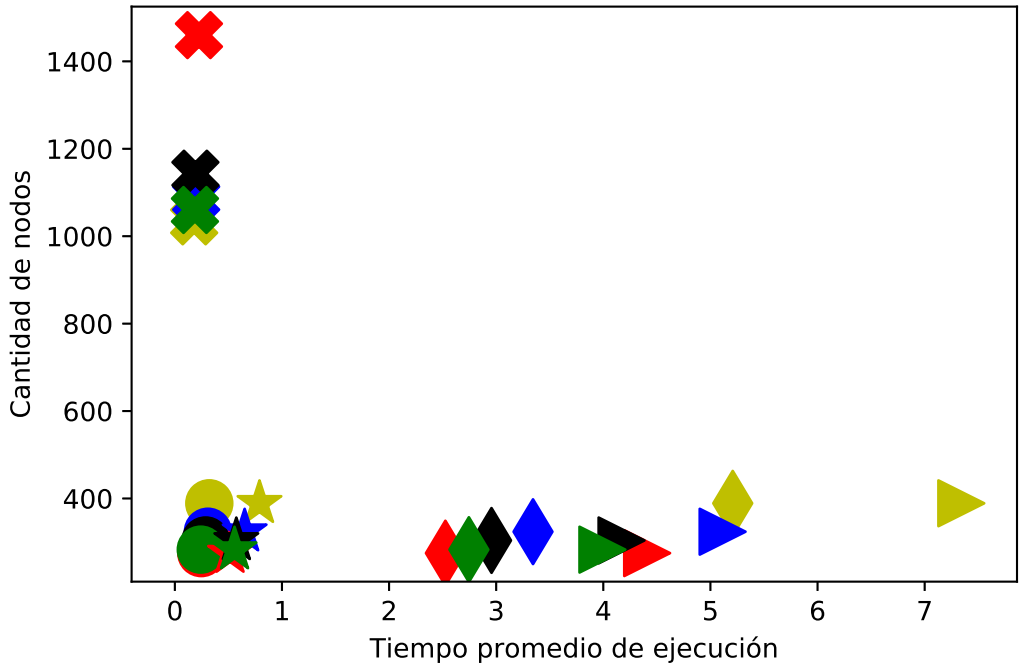


Figura 2: Gráfica de dispersión. Número de nodos respecto al tiempo de ejecución.

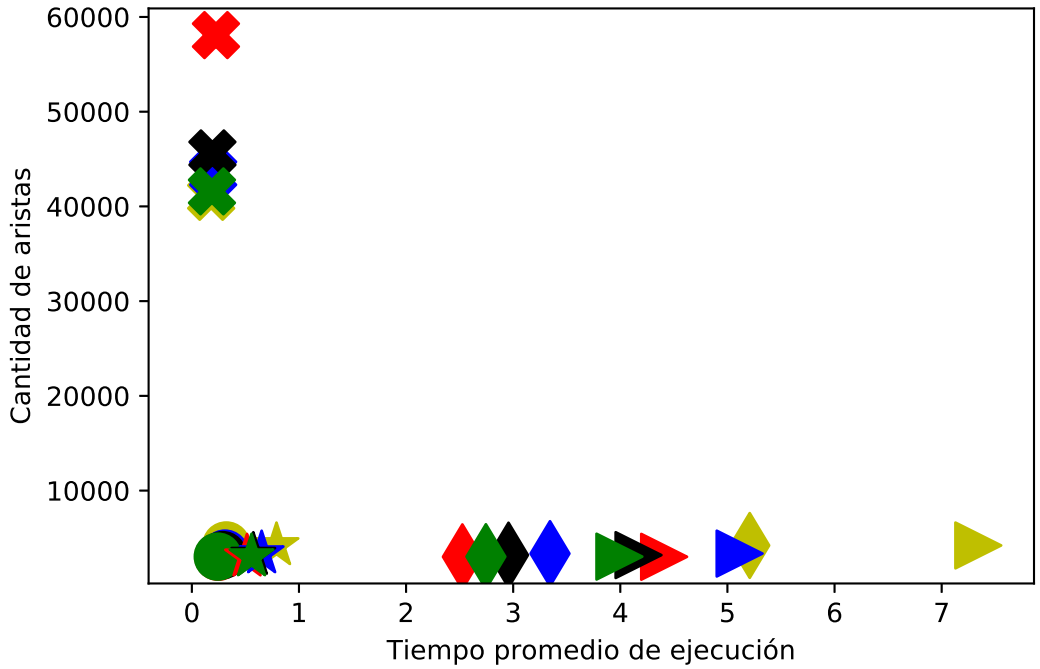


Figura 3: Gráfica de dispersión. Número de aristas respecto al tiempo de ejecución.

Referencias

[1] Matplotlib. urlhttps://matplotlib.org. Accedido 18-3-2019.

[2] Networkx. https://networkx.github.io/documentation/stable/reference/algorithms/.

[3] Python data analysis library. urlhttps://pandas.pydata.org, 2018). Accedido 18-3-2019.