Complejidad asintótica

4166

2 de abril de 2019

Introducción

Distintos algoritmos se pueden usar para resolver los mismos problemas; el desempeño de los algoritmos es la manera de evaluar a los mismos en base a su eficiencia en la resolución de dichos problemas. Existen distintos métodos para evaluar del desempeño, esto se debe al tipo de algoritmos y sus problemas. En este documento se verá que afecta directamente el desempeño de ciertos algoritmos de `Networkx`.

1. Especificaciones técnicas

La computadora en la que se corrió los algoritmos es una Macbook Pro, cuyo procesador es un Intel Core i5 de 2.3 GHz. Tiene una memoria RAM de 8 GB 2133 MHz.

2. Algoritmos

Los algoritmos a implementar son los siguientes.

`maximum_flow_value()`

Este algoritmo recibe como entrada un un grafo no orientado con capacidad en sus aristas junto con un par de nodos, el nodo *fuente* que es de donde parte el flujo y el nodo *sumidero* que es a donde llega el flujo. Devuelve el valor del flujo máximo soportado entre los nodos fuente y sumidero.

Utiliza el método de la etiqueta mas alta, es decir, de tomar el camino por el cual hay mas capacidad de aumento de flujo.

`flow.edmonds_karp()`

Este algoritmo recibe como entrada un grafo con capacidad junto con los nodos fuente y sumidero; lo que devuelve es la *red residual* después de haber calculado el flujo máximo.

Tiene rasgos parecidos al algoritmo de Ford-Fulkerson pero la diferencia más significativa es que el algoritmo de Edmonds Karp busca el camino mas corto para encontrar una ruta de aumento, es decir, explora todos los caminos posibles entre la fuente y el sumidero buscando cuales tienen capacidad disponible.

`flow.boykov_kolmogorov()`

Este algoritmo devuelve la red residual resultante después de calcular el flujo máximo, utilizando el algoritmo Boykov-Kolmogorov.

3. Generadores

Los generadores de grafos implementados fueron los siguientes.

`dense_gnm_random_graph()`

Este generador de grafos toma como parámetros la cantidad de nodos y aristas que se desean obtener; y devuele un grafo de entre los posibles grafos que cumplen con esos parámetros.

`random_graphs.barabasi_albert_graph()`

Este comando genera grafos aleatorios agregando las aristas de manera preferencial. Al generador se le dan los parámetros n y m que representan los nodos totales y el número de aristas a unir desde un nuevo nodo a los nodos existentes de acuerdo al algoritmo de Barabási-Albert.

`duplication.duplication_divergence_graph()`

A este generador se le dan los parámetros n y p que representan el total de nodos y la probabilidad de retención de aristas incidentes a los nodos originales.

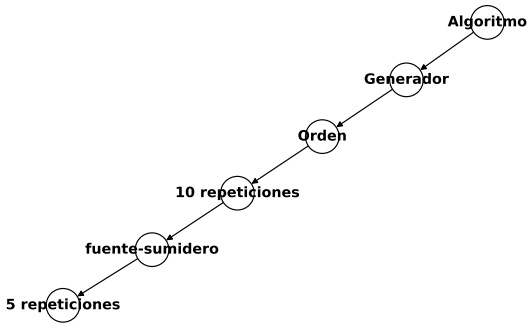

Figura 1: Diagrama de flujo aplicada en la metodología.

4. Metodología

El interés de este estudio es ver que variables influyen en el tiempo del algoritmo y de que manera influyen estas variables.

Cada algoritmo usó los tres generadores; cada generador hizo diez grafos con cuatro cantidades de nodos diferentes, cada grafo de orden logarítmico base cuatro; y se corrió el algoritmo cinco veces por cada par de nodo fuente-sumidero.

Como se muestra en la figura 1 son tres algoritmos, con tres generadores, cuatro ordenes y cinco diferentes pares de nodos fuente y sumidero.

La implementación en python se lleva a cabo con el código siguiente.

```
def correr(generator, algorithm, cardinalidad, l):
    if generator==0:
        G=AddEdges(nx.dense_gnm_random_graph(cardinalidad, int
    ((0.5*size*(size-1))-(1-((densidad+1)/10))*(0.5*size*(size-1)))
    ))
        nodes=RandNodes(cardinalidad-1,G)
        data[contador,3]=nx.density(G)
        if algorithm==0:
            nx.maximum_flow_value(G,nodes[l][0],nodes[l][1])
        elif algorithm==1:
            nx.algorithms.flow.edmonds_karp(G,nodes[l][0],nodes[l
    ][1])
        elif algorithm==2:
            nx.algorithms.flow.boykov_kolmogorov(G,nodes[l][0],
    nodes[l][1])
    elif generator==1:
        F=AddEdges(nx.generators.random_graphs.
    barabasi_albert_graph(cardinalidad, random.randint(1,
    cardinalidad-1)))
        nodes=RandNodes(cardinalidad-1,F)
        data[contador,3]=nx.density(F)
        if algorithm==0:
```

```
17                nx.maximum_flow_value(F,nodes[l][0],nodes[l][1])
18            elif algorithm==1:
19                nx.algorithms.flow.edmonds_karp(F,nodes[l][0],nodes[l
      ][1])
20            elif algorithm==2:
21                nx.algorithms.flow.boykov_kolmogorov(F,nodes[l][0],
      nodes[l][1])
22        elif generator==2:
23            J=AddEdges(nx.generators.duplication.
      duplication_divergence_graph(cardinalidad, random.random(),
      seed=None))
24            nodes=RandNodes(cardinalidad-1,J)
25            data[contador,3]=nx.density(J)
26            if algorithm==0:
27                nx.maximum_flow_value(J,nodes[l][0],nodes[l][1])
28            elif algorithm==1:
29                nx.algorithms.flow.edmonds_karp(J,nodes[l][0],nodes[l
      ][1])
30            elif algorithm==2:
31                nx.algorithms.flow.boykov_kolmogorov(J,nodes[l][0],
      nodes[l][1])
32
33 data=np.arange(1800*5, dtype=float).reshape(1800,5)
34 contador=0
35 for generador in range(3):
36     for algoritmo in range(3):
37         for orden in (16,64,256,1024):
38             size=orden
39             for densidad in range(10):
40                 for repeticion in range(5):
41                     tiempo_inicial=time()
42                     correr(generador, algoritmo, size, repeticion)
43                     tiempo_final=time()
44                     tiempo_ejecucion=tiempo_final-tiempo_inicial
45                     data[contador,0]=generador
46                     data[contador,1]=algoritmo
47                     data[contador,2]=orden
48                     data[contador,4]=tiempo_ejecucion
49                     contador+=1
```

Complejidad_experimental.py

5. Resultados

Con esa información se grafica la figura 2 cuya variable independiente son las observaciones, la variable dependiente es el tiempo que hizo en esa observación y cada punto representa una combinación de algoritmo y generador.

Se hizo una prueba de correlación entre las variables, para ver si las variables se relacionan entre ellas y con el tiempo de ejecución. Lo que muestra la figura 3 es que el orden y la densidad se relaciona directamente proporcional con el tiempo de ejecución.

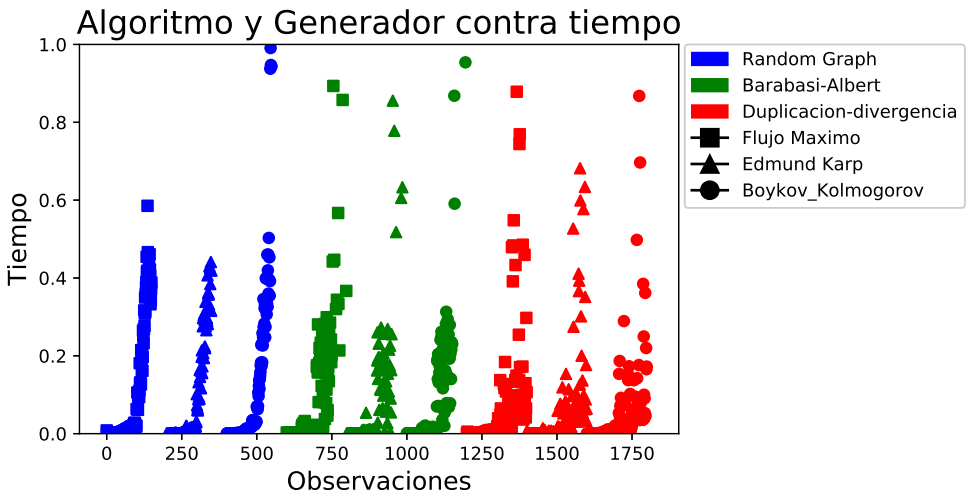

Figura 2: Los colores representan a los generadores y las figuras representan algoritmos.

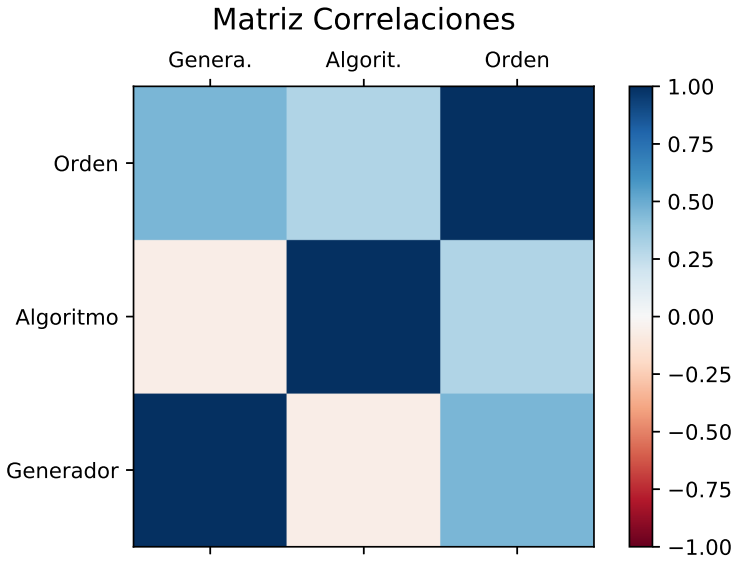

Figura 3: Correlaciones entre los datos cuantitativos.

Luego, se hizo un ANOVA para identificar si las medias de los conjuntos eran iguales o hay diferencias significativas en ellas. Se obtuvo los resultados que se muestra en la tabla llamada `ANOVA.txt` en el cual los valores de la tercer columna que son mayores a 0.1, sus medias son iguales y los valores menores 0.1 se rechaza la hipótesis de que las medias sean iguales.

`ANOVA.txt`

```
                                    sum_sq  ...         PR>F
Generador                         6.734005  ...  1.081764e-01
Algoritmo                        41.693816  ...  6.612653e-05
Orden                          2999.602915  ...  4.550752e-195
Densidad                        643.580915  ...  3.316542e-52
Generador:Algoritmo              22.285871  ...  3.500409e-03
Algoritmo:Orden                 122.376829  ...  1.001579e-11
Orden:Densidad                 1701.604229  ...  5.875774e-123
Generador:Orden                   7.490465  ...  9.022655e-02
Generador:Densidad               31.830308  ...  4.868091e-04
Algoritmo:Densidad              168.338921  ...  1.671812e-15
Generador:Algoritmo:Orden        49.356391  ...  1.429967e-05
Generador:Algoritmo:Densidad     54.666493  ...  4.985981e-06
Generador:Densidad:Orden         96.271378  ...  1.492816e-09
Algoritmo:Densidad:Orden        470.121502  ...  3.064382e-39
Generador:Algoritmo:Densidad:Orden  154.609626  ...  2.217023e-14
Residual                       4650.515320  ...           NaN

[16 rows x 4 columns]
```

Referencias

[1] NetworkX Developers, "Networkx documentation," 2012.

[2] A. C. Barrero, G. W. de García, and R. M. M. Parra, *Introducción a la Teoría de Grafos.* ELIZCOM SAS, 2010.

[3] J. M. Six and I. G. Tollis, "A framework for circular drawings of networks," in *International Symposium on Graph Drawing.* Springer, 1999, pp. 107–116.