Tarea 4 Optimización de Flujo de Redes

Evely Gutiérrez Noda

22 de abril de 2018

1. Introducción

En el siguiente reporte se abordará el tema estudiado en la clase de Optimización de Flujo de Redes. En dicha clase continuamos el estudio sobre la construcción de grafos, utilizando lenguaje Python [1] para la programación cuando queremos crear un grafo, además utilizamos la herramienta Gnuplot [2], la cual vinculamos con Python para representar el grafo antes programado.

Esta vez en la clase se realizaron experimentos para crear grafos circulares conectando sus aristas con una probabilidad **p** y en dependencia de un valor **k** para la cantidad de conexiones. Se trabajó además con dos funciones para calcular el promedio de las distancias de las densidades de las vecindades de los nodos y para calcular la densidad del grafo.

2. Descripción del trabajo en clase

Primeramente se creó un grafo partiendo de una cantidad de nodos **n**, estos nodos se conectaran con los otros en dependencia de un valor **k**, el cual indicará la cantidad de conexiones que tendrá un nodo y lo conectara con el que le sigue y así susecivamente. Los nodos se crearán partiendo de un centro definido (se tomó como centro**(0.1 y 0.5)**). Además se tuvo en cuenta la distancia de los nodos al centro por un radio **r (se tomó 0.5)**. A continuación se muestra el código de la función programada para la creación del grafo antes descrito.

```
def GuardaCirculo(self, dest, k, N, prob, r, xo, yo): #Para crear
    vertices que formen un circulo y guardarlos en un .txt
       self.destino = dest
       self.i = 1
       size = random()
       with open(self.destino , "w") as circulo:
           for n in range(N):
               xn = xo + r *(math.cos(2*math.pi * (n/N)))
               yn = yo + r *(math.sin(2*math.pi * (n/N)))
               print(xn, yn, file = circulo)
               self.x[n] = xn
               self.y[n] = yn
               self.agrega(size, (xn, yn))

           for a in range(N):
               for j in range(k):
```

```
            jj = (a+j+1)%n
            self.ConectaAristas(str(a),str(jj))
            self.i += 1

    for i in range(n-1):
        for j in range(n-2*k-1):
            jj = (i+j+k+1)%n
            if random() < prob:
                self.ConectaAristas(str(a),str(jj))

    return self.E
```

El grafo que quedó como resultado se muestra en la figura **??**.

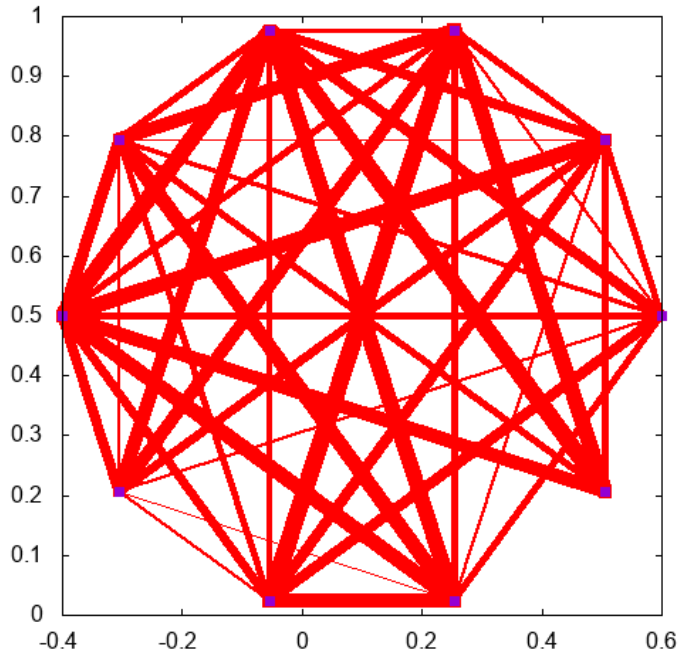

Figura 1: Grafo resultante.

A continuación, se muestra como quedaron las funciones **avgdist** y **clustcoef2** para calcular el promedio de la distancia, para la cual se trabajó con el algoritmo creado en la Tarea 2 [[3]] , **FloydWarshall**, y la otra función es para el calculo de la densidad del grafo.

```
def avgdist(self): #Promedio de las distancias
    self.d = self.FloydWarshall()
    self.sumatoria = sum(self.d.values())/ len(self.d)
    return self.sumatoria
```

```
def clustcoef2(self): # Densidad del grafo
```

```
        g = len(self.vecinos2) - 1
        valor = 0
        for v in range(1, g):
            m = 0
            for u in self.vecinos2[str(v)]:
                for w in self.vecinos2[str(v)]:
                    if u in self.vecinos2[str(w)]:
                        m+= 1
            n = len(self.vecinos2[str(v)])
            if n > 1:
                valor += m/(n*(n-1))
        return(valor/g)

    def ver(self):

        a = self.vecinos
        return a
```

Este trabajo con los algoritmos se realizó con varios grafos de distintos tamaños y algunos del mismo tamaño, se fue aumentando en dies en cada corrida el tamaño de los grafos, se midió el tiempo de ejecución para cada vez que se corrían los algoritmos. Se realizaron pruebas de corridas de hasta treinta veces con tamaños de grafos que llegaron hasta ciento cincuenta, y en cada una se almacenaron los tiempos de ejecución para cada función en ficheros de texto que se utilizaron para representar los resultados en un diagrama.

Se realizaron varias pruebas con varios grafos de distintas características para medir el tiempo de ejecución de estas funciones. Uno de los ejemplos se realizó con un grafo **G1** de tamaño inicial de nodos **(n)** igual a 90, y fue aumentando el tamaño de n en 10, para lograr los resultados de la figura 2.

Para grafos de igual tamaño, el experimento se realizó con grafos de sesenta nodos y se hicieron treinta corridas. Se guardaron los datos de los tiempos de ejecución de cada algoritmo en 5 veces que se realizó el experimento, y se graficaron los resultados de los tiempos en un diagrama que se muestra en la figura 3.

Referencias

[1] Python Software Fundation, `www.python.org/`

[2] Gnuplot, `www.gnuplot.info`

[3] Tarea2, `https://github.com/EvelyGutierrez/Optimizacion-de-flujo-de-redes/blob/master/Tarea2/VersionFinal/Tarea%202.pdf`

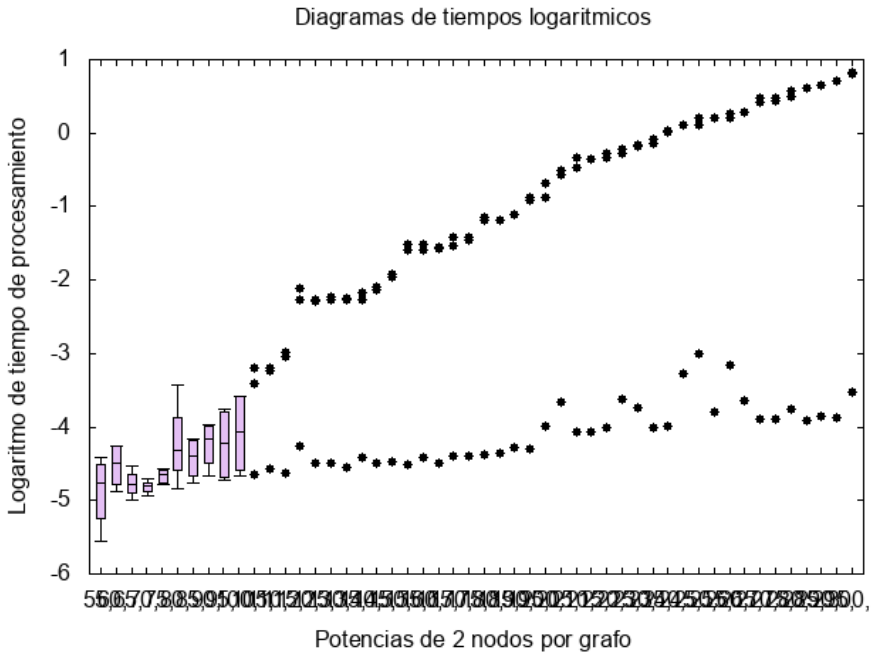

Figura 2: Diagrama de tiempos de ejecución.

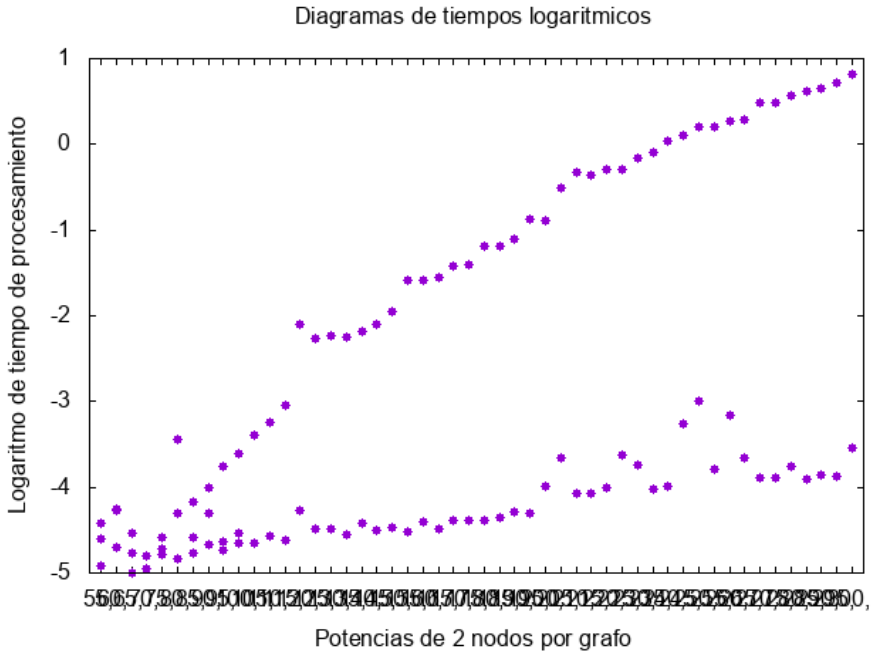

Figura 3: Diagrama de distancia contra probabilidad