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 conectará con el nodo 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 (se tomó como radio **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 1.

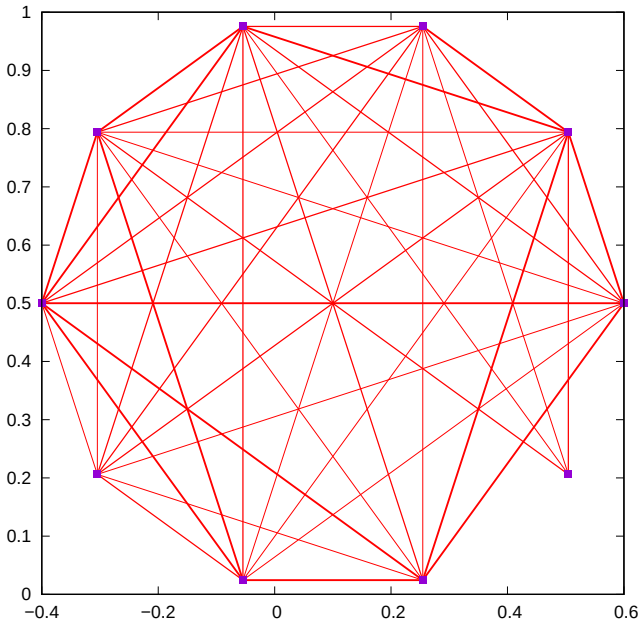

Figura 1: Grafo resultante.

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 cálculo de la densidad del grafo, la cual se calculó tomando el número de aristas entre el número de aristas máximos que podría tener un nodo, de modo que todos los nodos queden conectados con todos.

```
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 se realizó con varios grafos de distintos tamaños y algunos del mismo tamaño, se fueron aumentando en cinco en cada corrida el tamaño de **n** en los grafos, se midió el tiempo de ejecución para cada vez que se corrían las funciones creadas y los grafos construidos. Se realizaron pruebas de corridas de hasta diez veces con tamaños de grafos que llegaron hasta $\mathbf{n = 400}$, 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 que se muestra en la figura 2.

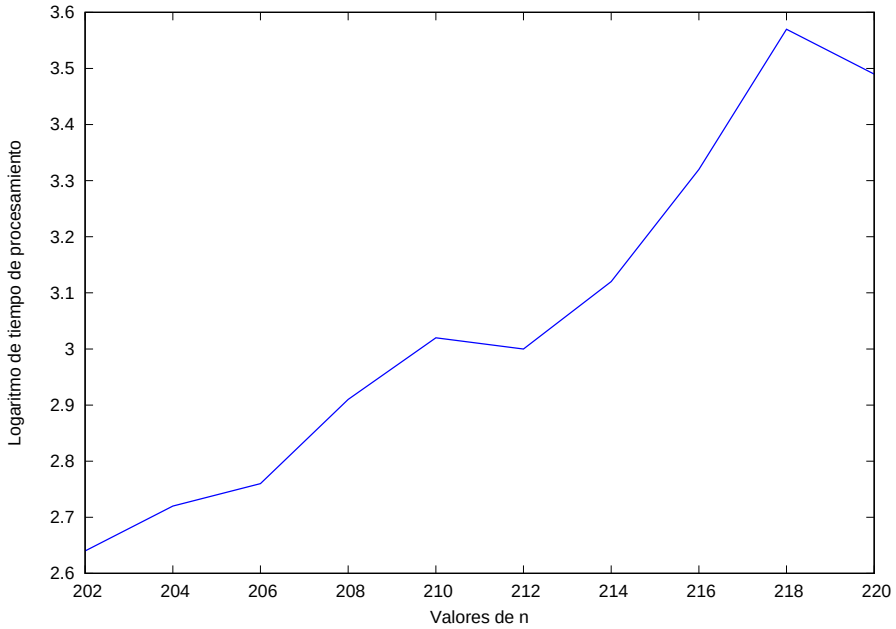

Figura 2: Diagrama de tiempos de ejecución.

Luego se realizaron mas experimentos trabajando con estas funciones anteriormente descritas y variando el parámetro **k** y la probabilidad **p**, ambos para establecer las conexiones entre los nodos, los valores de **k** variaron desde 1 hasta la cantidad de corridas que se hicieron, es decir, hasta diez en algunas ocaciones.

El gráfico de la figura 4, es un diagrama de dos ejes **Y** (Izquierdo y Derecho) [4], el **eje y (Izquierdo)** describe la variación de la distancia normalizada que se calculó por la función **avgdist**. En el **eje y (Derecho)** se grafica la variacón de la densidad promedio calculada por la función **clustcoef2** y en el **eje x** se encuentran las probabilidades con que se trabajó.

Los valores de **p** variaron desde 0.1 hasta 1, y los resultados de los valores que arrojaron las funciones para este experimento se muestra en los diagramas de las figuras 3 y 4.

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`

[4] Gráficos de 3 ejes, `http://gnuplot.sourceforge.net/docs_4.2/node292.html`

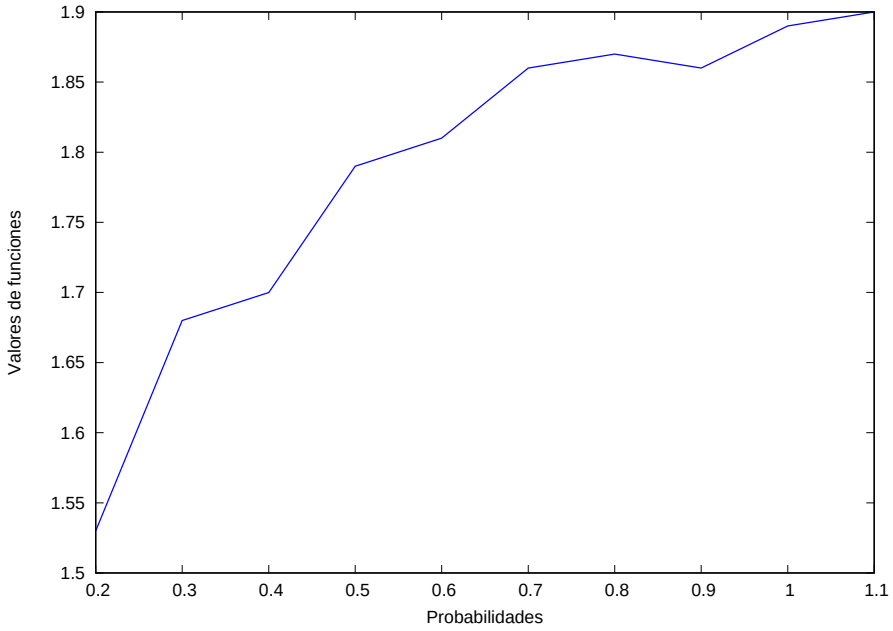

Figura 3: Diagrama de probabilidades contra funciones.

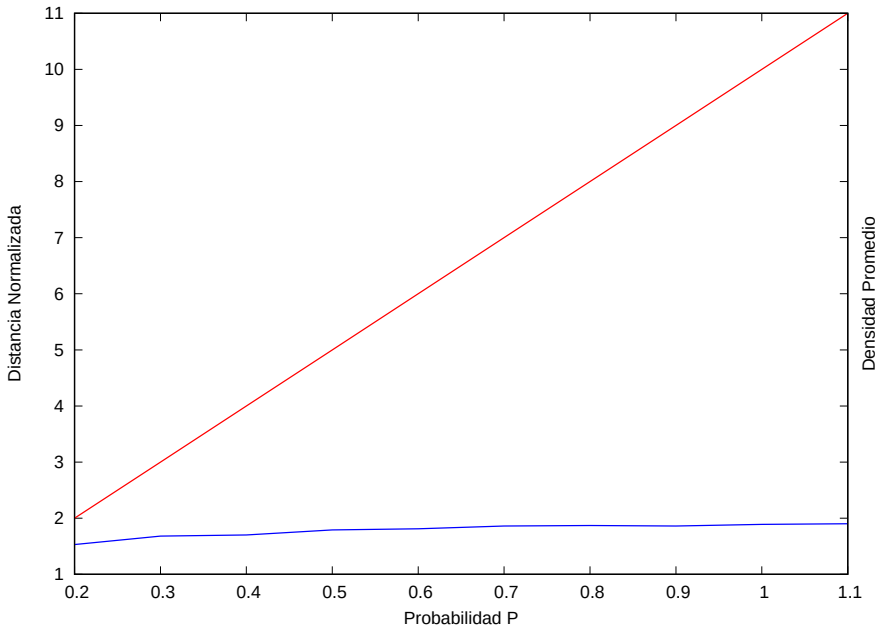

Figura 4: Diagrama de distancia contra probabilidad