Práctica 4

Jose David Martinez Ruiz

23 de abril de 2018

1. Introducción

En esta cuarta tarea lo que se busca es ver como es el comportamiento de la distancia mínima promedio en un grafo y también la conectividad promedio que el grafo tiene. Esto se hará en grafos cuyos nodos son colocados de forma circular y las conexiones que tiene depende de un número determinado de "pasos" que hay entre cada nodo. Nos interesa ver cómo influye una probabilidad de que un nodo se conecte con cualquier otro nodo en los coeficientes anteriormente mencionados. Además, veremos cómo se comportan los tiempos de ejecución de los algoritmos de Floyd-Warshall y Ford-Fulkerson con este tipo de grafos.

2. Desarrollo

Primero lo que se hizo fue cambiar la forma en la cual los grafos son creados, de manera que ahora sean, como lo llamaremos a partir de ahora, grafos circulares. Estos grafos circulares se crean dividiendo los 360 grados que tiene un circulo entre la cantidad de nodos que se van a colocar, luego de eso, utilizando coordenadas polares se coloca cada nodo en su correspondiente ángulo, el centro y el radio que se utilizaron fueron fijos.

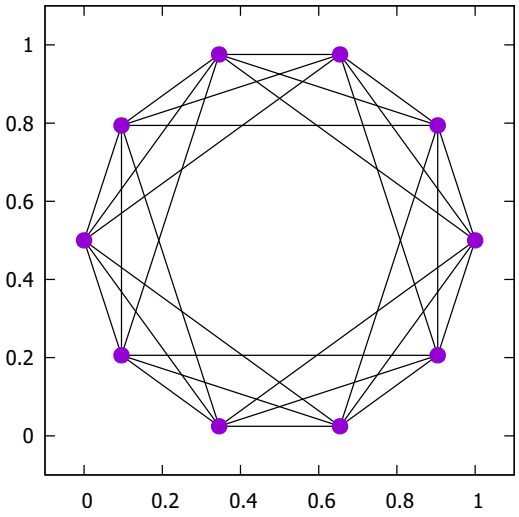
```
def crear(self, n):
        angulo = (2*pi)/n
        self.n = n
        for nodo in range(self.n):

            self.nodos.insert(nodo, ((0.5 + (0.5*cos(angulo*nodo))) ,
            (0.5 + (0.5*sin(angulo*nodo)))))
            self.x.insert(nodo, self.nodos[nodo][0])
            self.y.insert(nodo, self.nodos[nodo][1])
            self.vecinos[nodo] = set()
```

Ya con los nodos en el lugar que le corresponde, las conexiones entre ellos dependen de un numero entero k que determina los "pasos" que se da entre nodo y nodo. Por ejemplo, si k es 2, y hay 10 nodos numerados del cero al nueve, en ese caso el nodo cero se conectará con el nodo dos, el nodo uno con el nodo tres, y así sucesivamente. Además de eso también se conectarán dando pasos menores al dado, es decir si k es 2, se conectarán dando dos pasos y también un paso. Se harán conexiones con todos los enteros positivos menos o iguales al k dado.

```
def conexiones(self, k, prob):
        t = 0
        for r in range(k):
            for i in range(self.n):
                if i < (self.n - (r+1)):
                    self.A.insert(t,(self.x[i], self.y[i],
                    self.x[i+(r+1)], self.y[i+(r+1)]))
                    self.P.insert(t, (r+1))
                    self.vecinos[i].add(i+(r+1))
                    self.vecinos[i+(r+1)].add(i)
```

```
            t+=1
        else:
            self.A.insert(t, (self.x[i], self.y[i],
            self.x[(i + (r+1)) - self.n],
            self.y[(i + (r+1)) - self.n]))
            self.P.insert(t, (r+1))
            self.vecinos[i].add((i + (r+1)) - self.n)
            self.vecinos[(i + (r+1)) - self.n].add(i)
            t+1
```

Aquí se muestra un grafo con diez nodos y con un k igual a tres



Con el grafo ya terminado, con una probabilidad dada, un nodo de conecte aleatoriamente con otro, verificando antes de eso que la conexión no sea entre el mismo nodo ni tampoco que se haga una conexión anteriormente creada.

Como nos interesa también la implementación de los algoritmos usados en la tarea anterior, necesitamos darles pesos a las aristas. En este caso los pesos de las aristas están dados por la cantidad de "pasos" que se dio para llegar de un nodo a otro. Para aquellas aristas que se crearon de forma aleatoria se ajustó de manera que el peso que tenga sea la más pequeña. Por ejemplo, supongamos que tenemos diez nodos, y se crea una arista que conecte al nodo cero con el nodo siete, en lugar de que tenga un peso de siete, tendrá un peso de tres.

Ya con todo listo se empezaron a sacar los datos que necesitamos. Para poder sacar la distancia mínima promedio, lo que se hizo fue utilizar el algoritmo de Floyd-Warshall que nos dice la distancia más pequeña que hay en cada par de nodo, se sumaron todas esas distancias y al final se dividió entre la cantidad total que había para así sacar el promedio de esas distancias. Para sacar el coeficiente de conectividad promedio lo que se hace es que para cada nodo se verifican sus vecinos, ya con sus vecinos identificados se van a contar cuales son vecinos entre sí, una vez con ese número se dividirá entre la cantidad de vecinos que tiene ese nodo multiplicado por esa cantidad de vecinos menos uno, esto se hará siempre y cuando el nodo tenga más de un vecino. Ese procedimiento se hará para cada nodo y se sumará el resultado de cada uno para al final dividirlo entre la cantidad total de nodos y así obtener el coeficiente de conectividad promedio, dicho coeficiente tendrá un valor entero cero y uno.

Esas cantidades son las que nos interesa ver cómo se comportan al cambiar la probabilidad de conexiones aleatorias, pero el coeficiente de conectividad promedio es un valor entre cero y uno mientras que la distancia mínima promedio es más grande que uno. Por eso hay que obtener una cota superior para esta distancia mínima promedio para así dividirla y obtener un valor entre cero y uno. Se realizaron pruebas de que valor tenía la distancia mínima promedio con probabilidad igual a cero, al hacer esto se obtiene un valor más grande de la distancia mínima promedio que

las que se obtuvieron al agregarles una probabilidad, esto se debe a que si no hay una conexión aleatoria, la distancia para llegar de un nodo a otro es más grande, en cambio si hubiera alguna conexión aleatoria la distancia para llegar de un nodo a otro es más pequeña debido a que habría una forma más fácil y directa de llegar. Por lo tanto, se empezó con una cota superior igual a la distancia minina promedio sin que hubiera conexiones aleatorias. Al realizar más pruebas se encontró una relación que hay entre esa distancia minina promedio y el número total de nodos, se encontró que la distancia mínima promedio es ligeramente menor que el número total de nodos divido entre dos, por lo que la cota superior que utilizamos es igual al número total de nodos divido entre dos.

Ya con todo lo anterior listo, se procedió a graficar los valores obtenidos para cada caso. El eje x representa la potencia a la que se eleva el numero dos, cuyo resultado será la probabilidad de que haya una conexión aleatoria. El eje y representa el valor entre cero y uno que se obtiene tanto para la distancia mínima promedio dividida entre la cota y el coeficiente de conectividad promedio. La linea de color rojo representa los resultados de la distancia mínima promedio dividida entre su cota, y la linea de color azul representa el coeficiente de conectividad promedio.

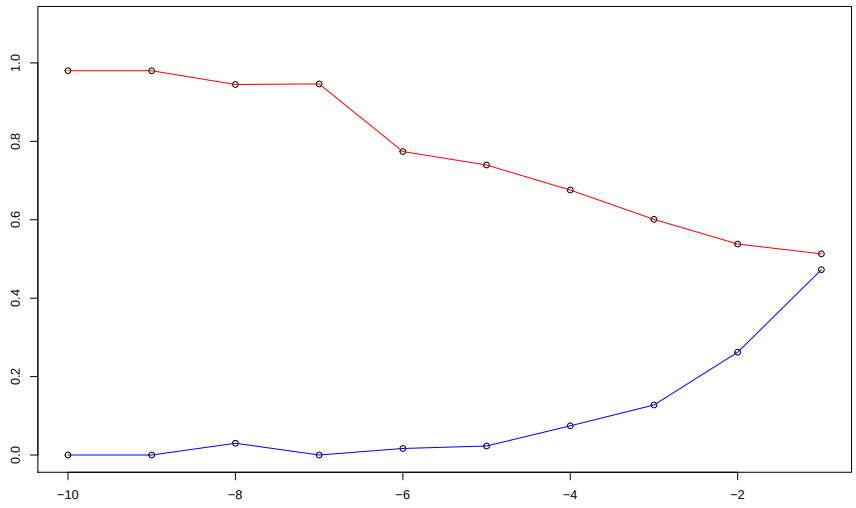


Figura 1: Valor k igual 1.

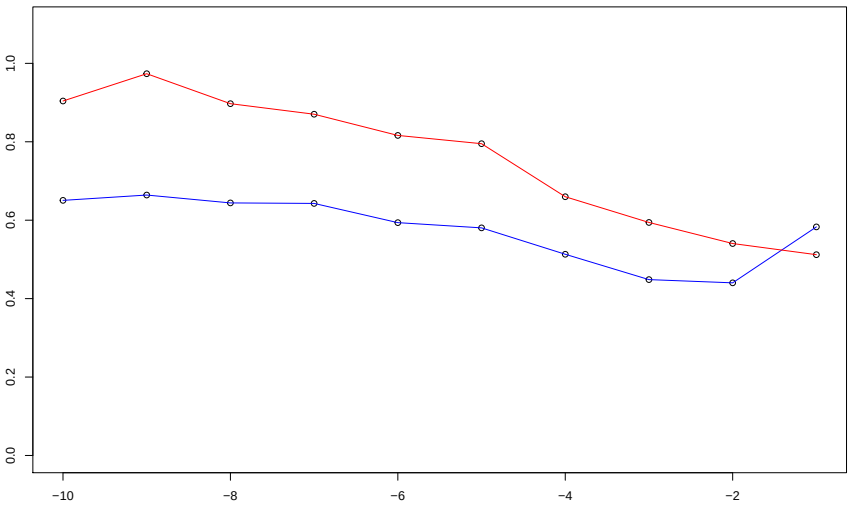


Figura 2: Valor k igual 5.

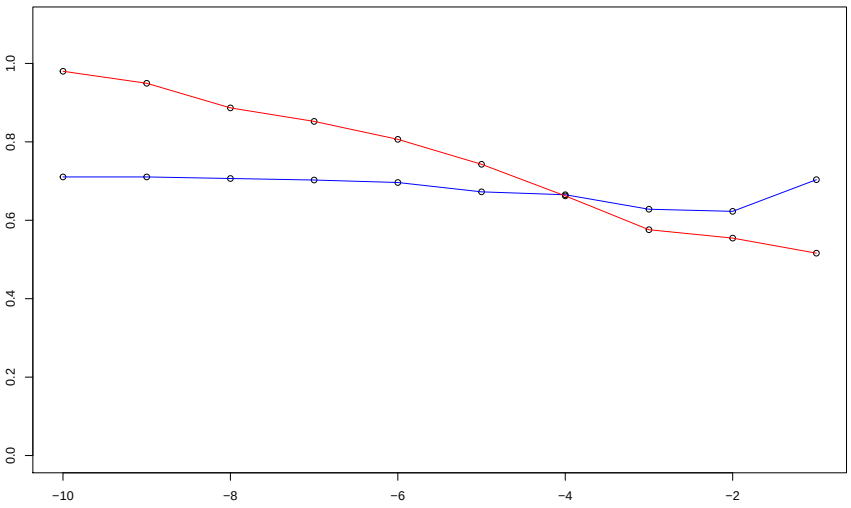


Figura 3: Valor k igual 10.

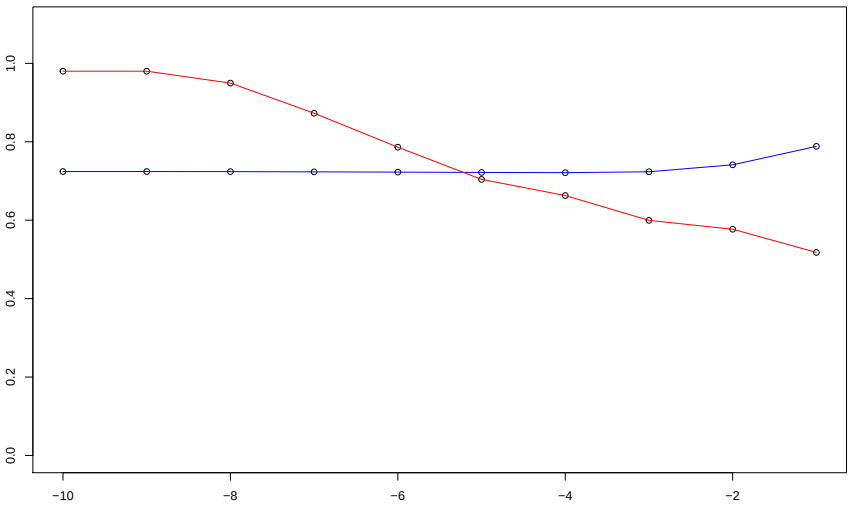


Figura 4: Valor k igual 15.

Habiendo visto ya el comportamiento de esos valores cuando la probabilidad varía y los "pasos.entre los nodos también, se procedió a ver el comportamiento de los tiempos de ejecución tanto para el algoritmo de Floyd-Warshall y el algoritmo de Ford-Fulkerson. Éstos se ejecutaron con grafos sin conexiones aleatorias y dando la mitad de la cantidad total de nodos como valor de k. El eje y de las cajas de bigote representa en segundos el tiempo de ejecicion de los algortimos, mientras que el eje x representa la cantidad nodos que tenía el grafo. Las cajas de bigote se muestran a continuación

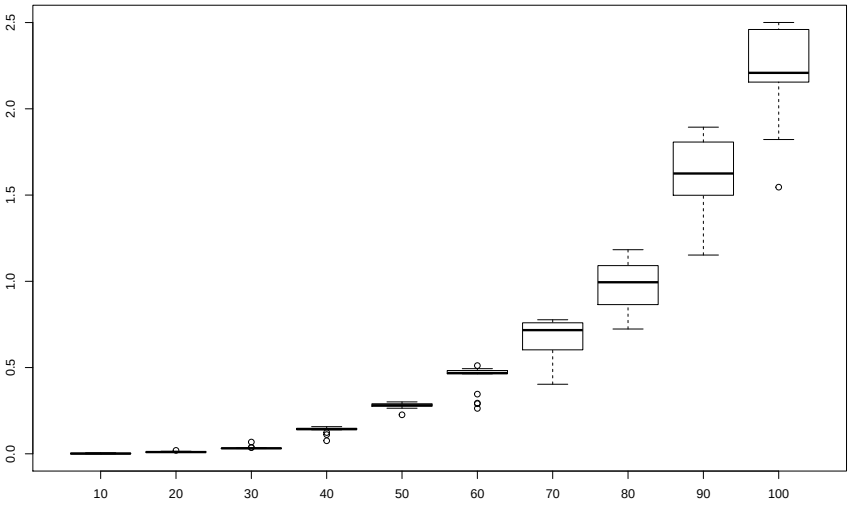


Figura 5: Algoritmo de Floyd-Warshall.

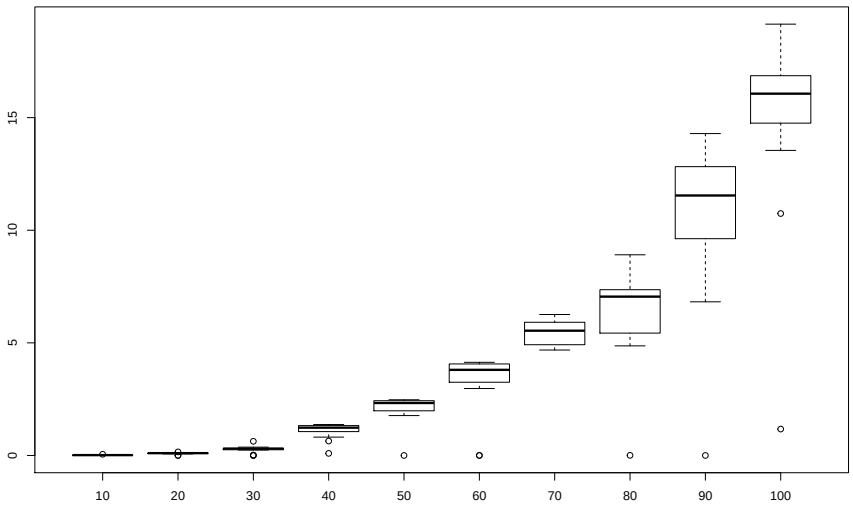


Figura 6: Algoritmo de Ford-Fulkerson.

3. Conclusiones

El coeficiente de conectividad promedio mientras más grande era el valor de k, éste se mantenía constante aunque la probabilidad de conexiones aleatorias fuera más grande, esto puede deberse a que con una k muy grande, ya hay muchas conexiones por lo que aunque haya conexiones nuevas, el coeficiente no se ve afectado por ellas. La distancia mínima promedio dividida entre su cota, en cada valor de k presentaba un comportamiento similar, mientras más grande fuera la probabilidad de conexiones aleatorias, la distancia mínima promedio disminuía debido a que con las conexiones nuevas, el algoritmo de Floyd-Warshall podía encontrar distancias más pequeñas entre cada nodo.

Al igual que en la tarea anterior, la complejidad computaciones de los algoritmos utilizados es polinomial $O(n^3)$, ademas de que, en comparación con los grafos utilizados en la tarea anterior, con los grafos circulares los algoritmos más rápido.

Referencias

David Martinez. Tarea 2 Optimzación de flujo en redes. Marzo 2018. https://github.com/DavidM0413/Flujo-en-Redes/blob/master/practica3/p3