Práctica 5

Jose David Martinez Ruiz

7 de mayo de 2018

1. Introducción

En esta quinta tarea lo que se busca es ver cómo afecta el quitar nodos o quitar aristas en el flujo máximo que puede pasar de un nodo en específico a otro. Además de eso se busca ver también como afecta esto al tiempo de ejecución del algoritmo de Ford-Fulkerson. Se espera que lo tiempos del algoritmo se vaya disminuyendo conforme más nodos o aristas sean eliminados del grafo.

2. Desarrollo

Lo primero que se hizo en esta tarea fue cambiar la forma en la que los grafos son creados, en la tarea anterior teníamos grafos circulares, ahora tendremos, lo que llamaremos a partir de ahora, grafos cuadrados. Estos grafos cuadrados se crean a partir de un valor entero k que representa la cantidad de nodos que habrá a lo largo y a lo ancho, de manera que haya un total de k^2 nodos. Al colocar los nodos de esta forma, se crea una malla cuadrada donde los nodos están colocados de manera simétrica.

```
def crear(self, k):
        self.n = k*k
        t = 0
        for i in range(k):
            for j in range(k):
                self.nodos.insert(t, ((1/(k-1))*j, (1/(k-1))*i))
                self.coord[t] = (j,i)
                self.x.insert(t, self.nodos[t][0])
                self.y.insert(t, self.nodos[t][1])
                t+=1
```

Los nodos se conectarán a partir de un valor entero l, este valor hace que todos los nodos cuya distancia Manhattan sea menor o igual que l se conecten entre sí. La distancia Manhattan entre dos puntos se obtiene al sumar el valor absoluto de la diferencia de los componentes horizontal y vertical de los puntos.

```
def Manhattan (p1, p2):
    p = (abs(p1[0] - p2[0]), abs(p1[1] - p2[1]))
    x = p[0] + p[1]
    return(x)
```

Para poder utilizar el algoritmo de Ford-Fulkerson es necesario que las aristas tengan un peso, las aristas que se crean tienen un peso generado aleatoriamente con una distribución normal con media de cinco y desviación estándar de raíz cuadrada de cinco. Se hizo esto para que las aristas tengan un peso lo más cercano posible a valores entre uno y diez.

```
def conexiones(self, l, p):
        self.L = l
        k = self.n**(1/2)
        for i in range(self.n):
```

```
        for j in range(self.n):
            if Manhattan(self.coord[i], self.coord[j]) <= l:
                self.A.append((self.x[i], self.y[i],
                self.x[j], self.y[j]))
                self.P.append(math.ceil(
                abs(normalvariate(5, (5**.5)))))
```

En la Figura 1 se muestra un ejemplo de un grafo cuadrado.

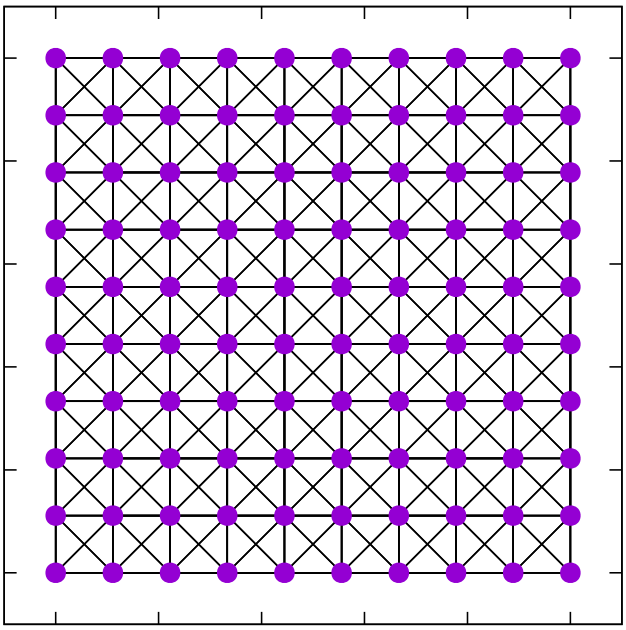

Figura 1: Valor k igual a 10, valor l igual a 2.

Ya con todo lo anterior listo, ya podemos crear grafos cuadrados conexiones dependientes del parámetro l. Queremos que ahora se generen aristas aleatorias que conecten cualquier par de nodos, para eso, dada una probabilidad p dada, se genera un número aleatorio y si éste resulta ser más pequeño que la probabilidad dada, se creará una arista aleatoria que no esté ya en nuestro conjunto de aristas.

```
    for i in range(self.n):
        for j in range(self.n):
            if random() < p:
                if (self.x[i], self.y[i],
                self.x[j], self.y[j]) not in self.A:
                    self.A.append((self.x[i], self.y[i],
                    self.x[j], self.y[j]))
                    self.P.append(math.ceil(
                    abs(normalvariate(5, (5**.5)))))
```

Debido a que nos interesa eliminar tanto nodos como aristas se tuvieron que crear funciones que se encargaran de eso. Para borrar nodos lo que se hizo fue que se generara un número aleatorio entre cero y el tamaño de nuestro conjunto de nodos para que así se escoja uno de ellos al azar, debido a que nos interesa saber el flujo entre un par de nodos, ese par de nodos no podrá ser eliminado. Ya sabiendo qué nodo vamos a eliminar se procede a verificar en nuestro conjunto de aristas y todas aquellas aristas que tengan al nodo serán eliminadas, además se eliminaran los pesos de esas aristas de nuestra lista de pesos.

```
def delnodo(self):
        n = len(self.x)
        k = math.floor(random()*n)
        if self.x[k] is not self.x[0] and self.y[k] is not self.y[(n-1)]
        and self.y[k] is not self.y[0] and self.x[k] is not self.x[(n-1)]:
            nodo = [self.x[k], self.y[k]]
            del self.x[k]
            del self.y[k]
            agh = []
            for i in range(0,len(self.A)):
                if (nodo[0] is self.A[i][0] and nodo[1] is self.A[i][1])
                or (nodo[0] is self.A[i][2] and nodo[1] is self.A[i][3]):
                    agh.append(i)
            agh.sort(reverse = True)
            for j in agh:
                del self.A[j]
                del self.P[j]
```

Para poder eliminar aristas, se hizo algo similar a lo que se hizo para eliminar nodos, se generó un número aleatorio entre cero y el tamaño total de aristas, para así obtener el índice de la arista que queremos eliminar de nuestra lista de aristas. Se hicieron pruebas de esta función y como afectaba quitar una arista al flujo de nuestro grafo, se observó que no había un efecto significativo en el flujo al quitar una arista, además de que el grafo con el que estamos trabajando es de cien nodos ($k = 10$) por lo que quitar una arista hasta que ya no haya flujo tomaría demasiado tiempo, por lo que se cambió la función para que quitara varias aristas cada vez que se llame la función y no solo una.

```
def delarista(self):
        b = 15
        for i in range(b):
            n = len(self.A)
            k = math.floor(random()*n)
            if k < len(self.A):
                del self.A[k]
                del self.P[k]
```

Con las funciones para borrar tanto nodos como aristas se procedió a ver cómo se comporta el algoritmo de Ford-Fulkerson mientras se quitan nodos o aristas hasta que ya no haya flujo entre esos dos nodos. Los nodos con los que vamos a trabajar son el nodo que se encuentra en la esquina inferior izquierda del grafo y el que se encuentra en la esquina superior derecha. Por cómo está construido el grafo, es obvio que siempre habrá flujo desde el nodo inicial hasta el nodo final, por lo que ahora veremos cómo quitar nodos y aristas afecta al flujo máximo entre esos dos nodos y tiempo que tarda en encontrarlo.

Se hicieron pruebas con grafos de cien nodos, con una probabilidad de aristas aleatorias de 0.0003 y se varió el valor del parámetro l desde uno hasta tres. Para cada uno de estos valores se hicieron cinco réplicas, tanto para el caso donde sólo eliminamos nodos como el caso donde sólo eliminamos aristas. En cada caso se eliminaron nodos o aristas hasta que ya no hubiera flujo entre los dos nodos.

De una de las réplicas obtenidas las figuras 2, 4, 6, 8, 10 y 12 muestran cómo se comporta el flujo máximo entre dos nodos mientras se eliminan nodos o aristas con diferentes valores de l. Las figuras 3, 5, 7, 9, 11 y 13 muestran cómo se comportan los tiempos de ejecución del algoritmo de Ford-Fulkerson mientras se eliminan nodos o aristas con diferentes valores de l.

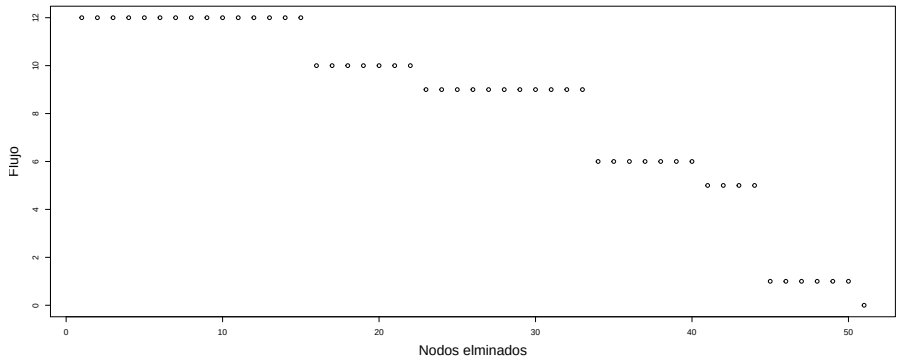

Figura 2: Eliminando nodos l igual 1.

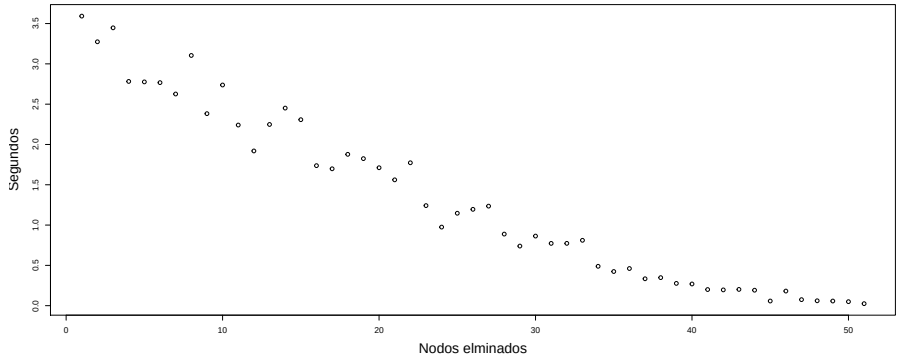

Figura 3: Eliminando nodos l igual a 1.

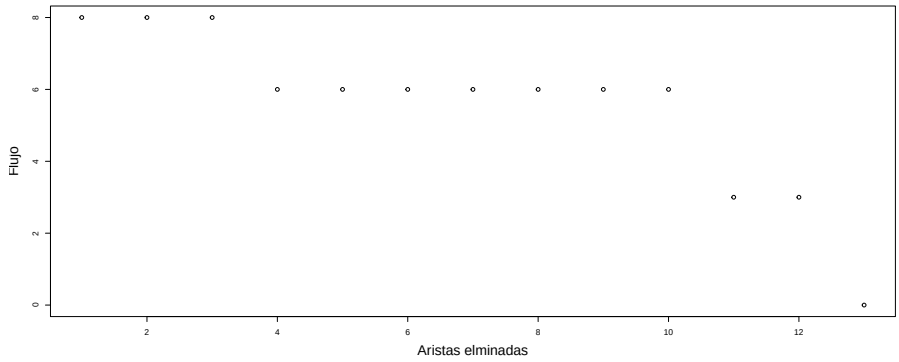

Figura 4: Eliminando aristas l igual a 1.

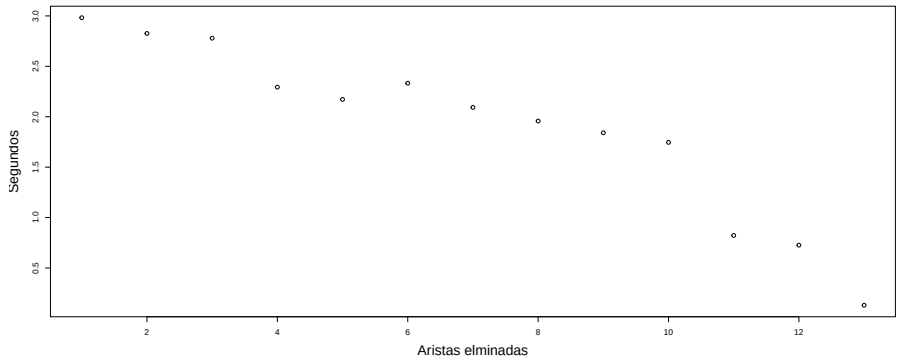

Figura 5: Eliminando aristas l igual a 1.

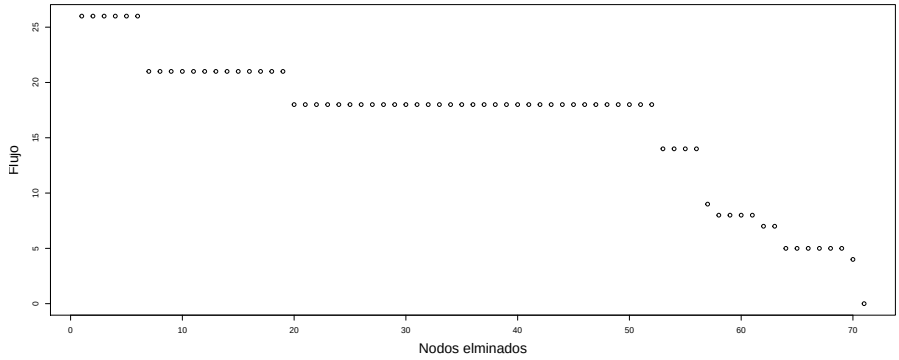

Figura 6: Eliminando nodos l igual a 2.

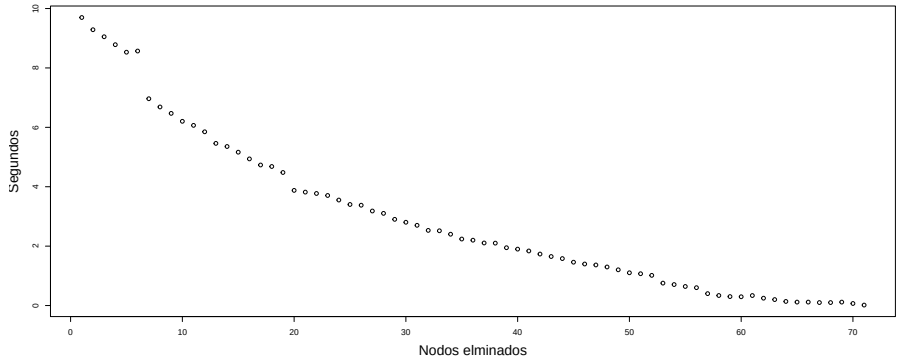

Figura 7: Eliminando nodos l igual a 2.

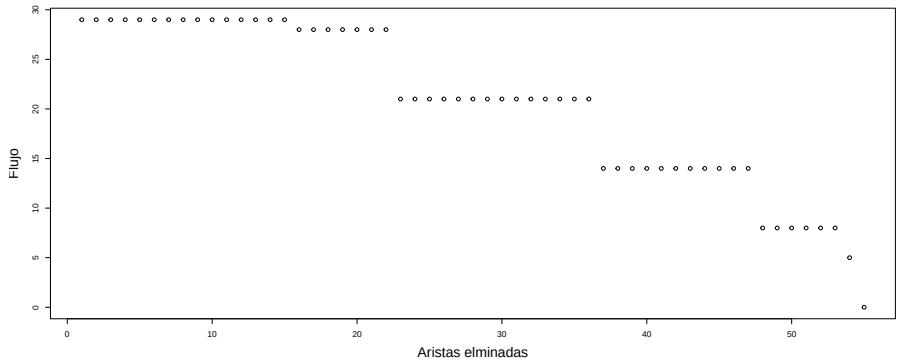

Figura 8: Eliminando aristas l igual a 2.

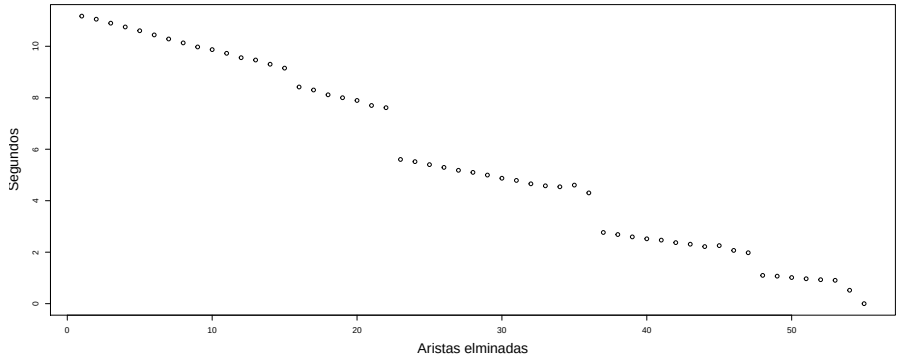

Figura 9: Eliminando aristas l igual a 2.

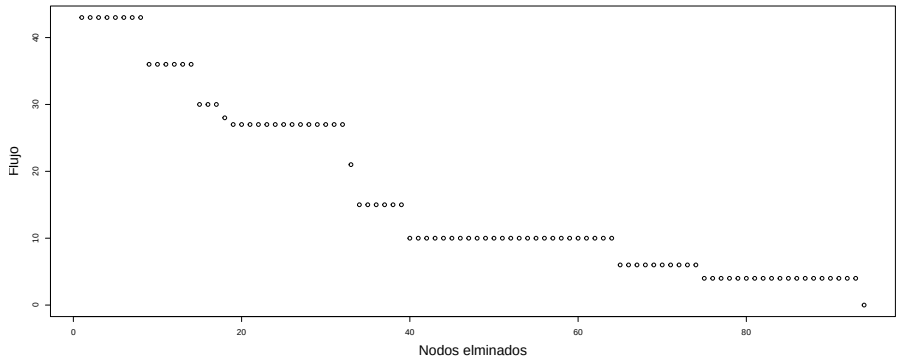

Figura 10: Eliminando nodos l igual a 3.

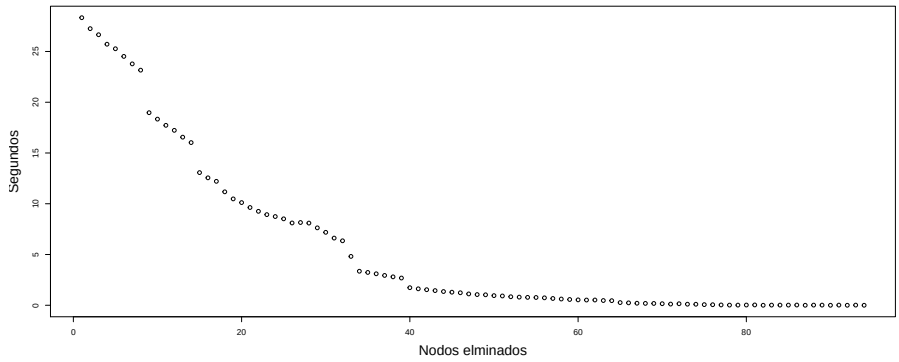

Figura 11: Eliminando nodos l igual a 3.

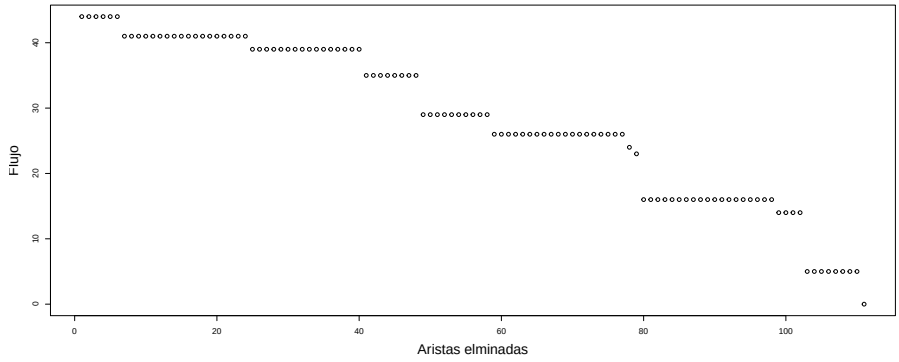

Figura 12: Eliminando aristas l igual a 3.

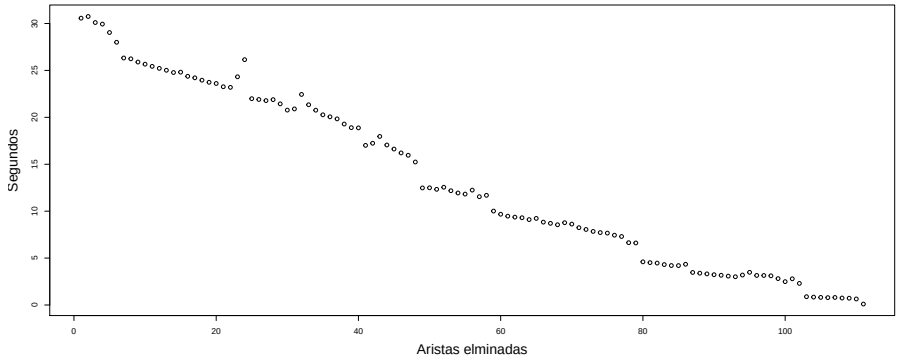

Figura 13: Eliminando aristas l igual a 3.

3. Conclusiones

Con valores de l pequeños la eliminación de nodos resulta menos efectiva que eliminar varias aristas al mismo tiempo, esto puede deberse a que con valores de l pequeños los nodos tienen menos cantidad de aristas que con valores de l más grande, en esos casos es más rápido eliminar nodos que varias aristas a la vez.

Con respecto a los tiempos de ejecución del algoritmo de Ford-Fulkerson, mientras más grande fuera el valor del parámetro l, el tiempo de ejecución del algoritmo de Ford-Fulkerson es mayor,

esto puede deberse a que con un valor de l grande, más aristas hay en el grafo, lo cual implica que buscar donde puede pasar flujo toma más tiempo. A la hora de verificar cuanto tiempo se tarda el algoritmo mientras se van eliminando tanto aristas como nodos, se obtuvo que después de haber eliminado muchos nodos o muchas aristas el algoritmo se ejecutaba más rápido.

Referencias

David Martinez. Tarea 4 Optimzación de flujo en redes. Abril 2018. https://github.com/DavidM0413/Flujo-en-Redes/blob/master/practica4/t4.py