Optimización de Flujo en Redes: Reporte 6

Mayra Cristina Berrones Reyes. 6291

Mayo 2018

Introducción

Como complemento de todos los reportes anteriores [1, 2] se llega finalmente a esta práctica, en la cual utilizamos diferentes partes de los temas que se vieron a lo largo del semestre para realizar una última experimentación.

En este caso, volvemos a retomar el tema de flujos máximos, y como tal, traemos de vuelta el algoritmo de Ford-Fulkerson [3, 4] para apoyarnos una vez más en él para darle forma a nuestros resultados.

Como se ha visto anteriormente [2] el problema más evidente al momento de utilizar el algoritmo de Ford-Fulkerson es que conforme se aumenta la cantidad de nodos en nuestro grafo, el tiempo de corrida del programa tiende a subir de manera bastante notoria. Este reporte se presenta como un pequeño experimento para ver otras formas en las que podemos buscar y conocer el flujo máximo sin necesidad de utilizar el algoritmo de Ford-Fulkerson. Al final se comparan los tiempos de procesamiento para saber si en realidad esto resulta eficiente.

Selección del grafo

Como primer paso se podía reutilizar cualquier tipo de grafo que ya se hubiera utilizado, pero en este caso se decidió hacer uno un tanto diferente. Como estamos hablando de flujos máximos, la manera más fácil de representarlo es si diferenciamos bien cual tenemos por nodo de entrada y cual tenemos por nodo de salida. En el reporte 5 [2] utilizamos un grafo en forma de malla, y la manera de identificar el nodo inicio y el nodo final fue ponerlos de diferente color. En este caso, se decidió colocarlos en lados opuestos de la gráfica, dejando el espacio entre ellos como lugar para poner los demás nodos.

Se retomó el tema de que los nodos pueden estar acomodados en cualquier parte de la gráfica (siempre y cuando no rebasen el nodo de inicio o el de final), y se les da sus coordenadas de manera aleatoria. Los pesos de las aristas también se manejan de manera aleatoria al igual que la probabilidad de que se unan los nodos.

```
#Extracto de código de clase Grafo()
def puntos (self, num):
      self.n = num
      self.nodos.append((-.3, 0.5, 0)) #Nodo de inicio
      #Listas Auxiliares
      self.aux.append((0))
      self.sec.append((0))
      #Nodos aleatorios
      for i in range(1, self.n - 1):
         self.nodos.append((random(), random(), i))
         self.aux.append((i))
```

```
            self.sec.append((i))
        self.nodos.append((1.3, 0.5, self.n -1)) #Nodo final o sumidero
        self.aux.append((self.n -1))
        self.sec.append((self.n -1))

    def conect (self, prob):
        for (x1, y1, u) in self.nodos:
            for(x2, y2, v) in self.nodos:
                if u is not v:
                    if random() < prob:
                    #Se utiliza la distancia para darle el valor a los pesos
                        de las aristas
                        self.aristas[(u,v)] = self.aristas[(v, u)] =
                            int(sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2) * 10)
                        self.A.append((x1, y1, x2, y2, u, v))
                        self.pesos.append(((sqrt((x2 - x1) ** 2 + (y2 - y1 )**
                            2)*10), (x1+x2)/2, (y1+y2)/2))
        print (self.aristas)
```

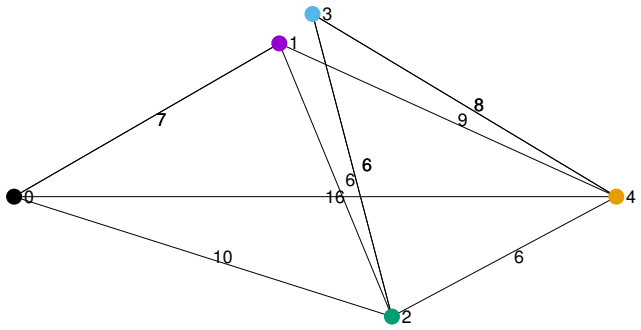


Figura 1: Para el número de nodos se seleccionó una cantidad de 5, para la probabilidad de conexión se le dio 0.4.

Descripción del experimento

Lo que se quiere lograr con esta práctica es, como se explicó en la introducción, lograr formar una herramienta que tenga un tiempo de procesamiento menor que el del algoritmo de Ford-Fulkerson, pero que a su vez tenga un nivel de exactitud aceptable para el flujo máximo del grafo.

En este caso lo que se lleva a cabo es que se toma una de las aristas de nuestro grafo de manera aleatoria, y con esto se toman las coordenadas de los nodos que la conectan, eliminando la arista y descartando el peso de ésta, haciendo que todos las aristas que estaban conectadas anteriormente al punto que se descartó, se conecten al punto al que este se unió. Al finalizar todas las uniones, en el grafo solo deben de quedar dos nodos unidos por una sola arista, que va a tener de peso la suma de todas las demás aristas que no se descartaron. Esto se espera que de cómo resultado un valor igual o cercano al valor que nos daría correr nuestro grafo con el algoritmo de Ford-Fulkerson.

Por la forma en que se desarrollo el grafo en esta práctica, existen restricciones que no permiten que el nodo de inicio y el nodo final terminen juntos.

```
#Extracto de código de clase Grafo()

   def merge(self):
      while len(self.aux) > 2: #Este ciclo va a correr hasta que solo
          queden dos nodos dentro de mi grafo.
         w = choice(self.aux)
         z = choice(self.aux)
         if w is not z: #No hay ninguna arista que se conecte un nodo
             con si mismo, por lo que se descarta
            if w is not self.nodos[0][2] and w is not
                self.nodos[len(self.nodos) - 1][2]:
            #Se prohibe tomar la arista que une al nodo de inicio con el
                nodo final.
               if w in self.aux:
                  if (z, w) in self.aristas:
                     del self.aristas[(z, w)]
                     del self.aristas[(w, z)]
                     self.sec[z] = str(self.sec[z]) +"#"+ str(self.sec[w])
                     self.aux.remove(w)

                     print (self.aux)
                     print (z, w)
                     for i in range(0, len(self.nodos) - 1):
                        if w is self.nodos[i][2]:
                           self.nodos.pop(i)
                           print(self.nodos)

                     for i in range(0, self.n):
```

```
                    if(i, w) in self.aristas:
                       if(z, i) in self.aristas:
                          self.aristas[(i, z)] =
                              (list(self.aristas.values())
                          [list(self.aristas.keys()).index((i, z))]) +
                              (list(self.aristas.values())
                          [list(self.aristas.keys()).index((i, w))])
                    #self.aristas[(i,z)] = self.aristas[(i,
                        z)].value() + self.aristas[(w, i)].value()
                    #self.aristas[(z, i)] =self.aristas[(z,
                        i)].value() + self.aristas[(w, i)].value()
                          self.aristas[(z, i)] =
                              (list(self.aristas.values())
                          [list(self.aristas.keys()).index((z, i))]) +
                          (list(self.aristas.values())[list(self.aristas.keys()).index((w,
                              i))])
                       else:
                          self.aristas[(i, z)] =
                              (list(self.aristas.values())
                          [list(self.aristas.keys()).index((w, i))])

                    #self.aristas[(i, z)] = self.aristas[(w, i)]
                          self.aristas[(z, i)] =
                              (list(self.aristas.values())
                          [list(self.aristas.keys()).index((w, i))])

                       print('estoy aqui')
                       print (w,i)
                       print(list(self.aristas.values())[list(self.aristas.keys()).index((i,
                           w))])

        print(self.nodos)
        print(self.sec)
        print (self.aristas)
        self.final =
            (list(self.aristas.values())[list(self.aristas.keys()).index((0,
            self.n -1))])
        print(self.final)
```

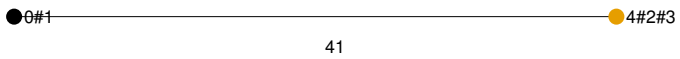


Figura 2: Al finalizar el ciclo de *while* dentro del código, la única arista que queda es la que une al nodo de inicio y al nodo final, y en esta se aprecia el peso final, que es la sumatoria de todas las aristas que no se descartaron.

Comparación

Como se menciona en la introducción de este reporte, lo que se busca es saber si nuestro experimento es una opción viable para no tener que usar el algoritmo Ford-Fulkerson. Para esto se realizaron varias corridas y se tomaron los datos pertinentes para realizar tablas.

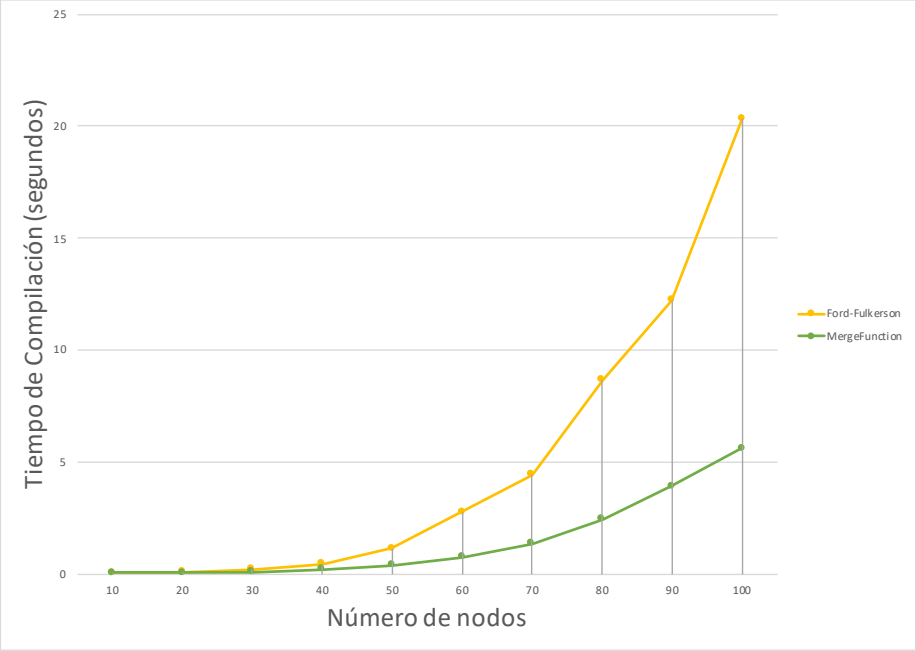
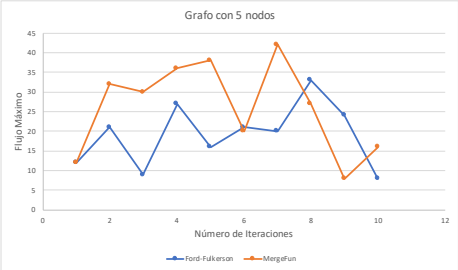
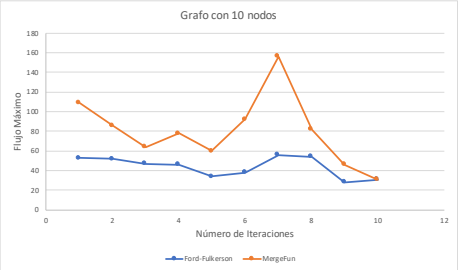


Figura 3: En esta tabla se ve la diferencia en el tiempo de procesamiento entre el Ford-Fulkerson y la función de *Merge* dentro del código.

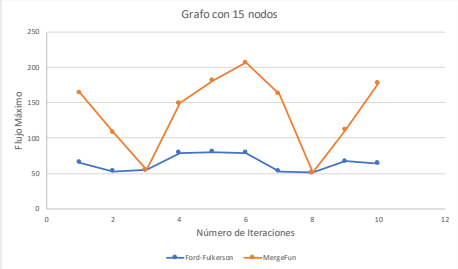
Estudiando esta tabla tal vez se puede asumir que resulta más conveniente utilizar el método de *Merge* ya que el método de Ford-Fulkerson se puede apreciar que va subiendo de tiempo a un nivel más rápido que el otro. Pero ahora podemos tomar en cuenta también la diferencia de flujo máximo entre una y otra función con las siguientes tablas.



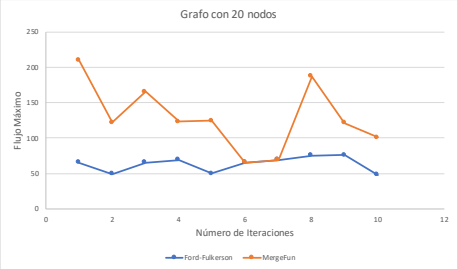
(a) Grafo con 5 nodos



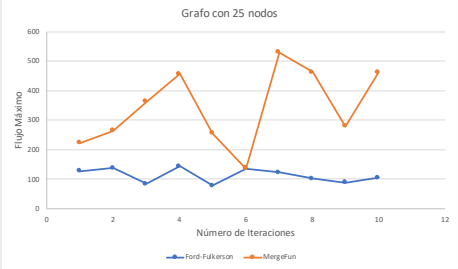
(b) Grafo con 10 nodos



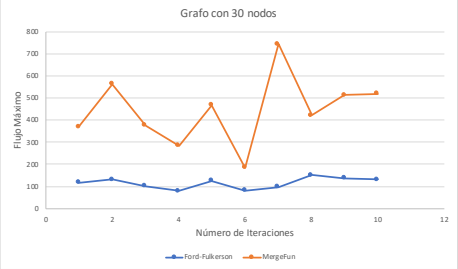
(c) Grafo con 15 nodos



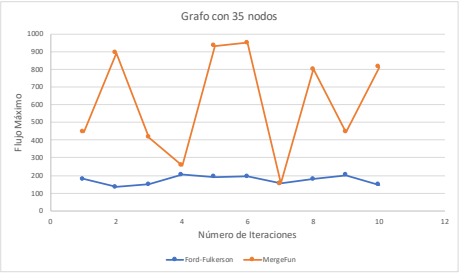
(d) Grafo con 20 nodos



(e) Grafo con 25 nodos



(f) Grafo con 30 nodos



(g) Grafo con 35 nodos

Figura 4: Gráficas donde se representa la diferencia en el resultado de flujo máximo en grafos de diferente cantidad de nodos

Conclusiones

Como conclusión podemos decir que comprobamos que nuestro método alternativo si es una buena sugerencia en cuanto al tiempo de procesamiento del programa, pero al mismo tiempo, como ya hemos visto en reportes anteriores, el algoritmo de Ford-Fulkerson es bastante exacto en cuanto al nivel máximo de flujo en redes, ya que utiliza la función de buscar todos los posibles caminos, lo cual es también la principal razón por la que el tiempo de procesamiento se va por las nubes cuando sube el número de nodos. A pesar de esto, podemos apreciar en nuestras últimas gráficas que nuestro método alternativo de unir nodos puede que no sea demasiado exacto al momento de sacar el flujo máximo. Como teoría podemos decir que esto puede deberse a la cantidad de aristas que desechamos, y con ellas la cantidad de flujo descartado.

Referencias

[1] Berrones Reyes, Mayra Cristina *Reporte4. Marzo 2018. https://github.com/mayraberrones94/Flujo-en-Redes/tree/master/Reporte4*

[2] Berrones Reyes, Mayra Cristina *Reporte5. Mayo 2018. https://github.com/mayraberrones94/Flujo-en-Redes/tree/master/Reporte5*

[3] Schaeffer, Elisa *https://elisa.dyndns-web.com/teaching/mat/discretas/md.html* Matemáticas Discretas. Grafos y árboles. **Consultado en Mayo 2018**

[4] Flows in Networks *Ford, L.R. and Fulkerson, D.R., kshttps://books.google.com.mx/books?id=fw7WCgAAQBAJ* **Consultado Mayo 2018**