# Optimización de Flujo en Redes: Distancia Manhattan

Alcantar Gómez Alan Arnoldo
Matricula: 1935040

7 de mayo de 2018

## 1. Introducción

El presente trabajo muestra la implementación de las distancias Manhattan para conectar los vértices de un grafo, cuyos pesos provienen de una distribución normal. Además, se agregaron uniones no presentes en el grafo de forma probabilista cuyos pesos son dados por una distribución exponencial. El objetivo del trabajo fue eliminar aristas y vértices de forma al azar, y observar sus efectos sobre los tiempos de procesamiento para obtener el flujo máximo utilizando el algoritmo *Ford Fulkerson*. Se trabajo con el lenguaje de programación *Python* y el graficador *Gnuplot*.

## 2. Teoría

La ***distancia Manhattan*** es una forma de geometría en la que la métrica usual de la geometría euclidiana es remplazada por una nueva métrica en la que la distancia entre dos puntos es la suma de las diferencias absolutas de sus coordenadas [2]. Dados dos vectores $\mathbf{p}$ y $\mathbf{q}$ en un espacio de dimensión $n$, la distancia Manhattan se define como:

$$D_{p,q} = \sum_{i=1}^{n} |p_i - q_i| \tag{1}$$

Sea $G(V, E)$ un grafo, con $V$ vértices, E aristas y donde por cada par de vértices $(u, v)$, tenemos una capacidad $c(u, v)$ y un flujo $f(u, v)$. Lo que se busca es maximizar el valor del flujo desde una fuente $s$ hasta un sumidero $t$. El algoritmo ***Ford Fulkerson*** inicia con $f(u, v) = 0$ para todo $(u, v)$ en $V$ y en cada iteración, se incrementa el flujo en G mediante el resultado de la busqueda del camino de aumento minimo que existe en la red residual $G_f$. El código se detiene cuando no existen más caminos aumentantes en la red residual $G_f$ y regresa el flujo máximo encontrado. En este caso las entradas que necesita el algoritmo son las aristas E del grafo y los puntos fuente y sumidero [3].

## 3. Descripción del algoritmos

Como era necesario que el vértice con la primera etiqueta estuviera en la esquina superior izquierda y que el nodo con la última etiqueta estuviera en la esquina inferior derecha, fue necesario asignar primero los nodos de la parte superior de izquierda a derecha, bajar un nivel y hacer de nuevo el recorrido de izquierda a derecha, como se muestra es las siguiente líneas de código.

```
for i in range(k-1,-1,-1):
   for j in range(k):
      self.nodos.add(q)
      self.coor.append((j,i))
```

En las siguientes las siguientes líneas de código se describe la idea para unir los vértices del grafo, esta se basa en la distancia Manhattan máxima $l$, dado un vértice, primero se verifica si existe otro vértice a una distancia $l$ a la derecha, después se verifica si existen dos vértices pero uno de ellos situado $l-1$ pasos a la derecha y 1 paso hacia arriba, y el otro vértice situado $l-1$ pasos a la derecha y 1 hacia abajo. De este modo a media que disminuyen los pasos en la horizontal aumentan los pasos hacia arriba y abajo, hasta llegar a $l$ pasos a la izquierda, de modo que la distancia Manhattan total sea $l$.

```
for i in self.nodos:
   for p in range(1,l+1):
      for j in range(p,-p-1,-1):
         x=self.coor[i][0]+j
         y_s=self.coor[i][1]+p-fabs(j)
         y_i=self.coor[i][1]+fabs(j)-p
         if ((x,y_s)) in self.coor:
            self.pesos[(i,self.coor.index((x,y_s)))]=int(random.normalvariate(6,1))
         if ((x,y_i)) in self.coor:
            self.pesos[(i,self.coor.index((x,y_i)))]=int(random.normalvariate(6,1))
```

Para agregar las aristas, se seleccionaron de forma aleatorio dos índices $i$ y $j$ que estuvieran dentro del rango de vértices, de modo que entre los índices obtenidos no existe una unión, y esta arista se agregan siempre y cuando cumplan de forma aleatoria se obtiene un valor menor que una probabilidad establecida.

```
for i in range(self.nn):
   for j in range(self.nn):
      if i != j:
         if random.random() <= p:
            if ((i,j)) not in self.pesos:
               self.pesos_p[(i,j)]=int(random.expovariate(0.1))+1
```

En el caso de eliminar vértices, se creo una lista ordenada de forma aleatoria de todos los vértices que forman el grafo excepto el inicio y final, de forma que al pasar por cada vértice se detectaba con que otro vértice formaba una arista y

esta era eliminada del grafo.

```
quitar_ver=random.sample(self.comodin,self.nn-2)
   for i in quitar_ver:
      for j in range(self.nn):
         if ((i,j)) in self.pesos:
            del self.pesos[(i,j)]
            del self.pesos[(j,i)]
         if ((i,j)) in self.pesos_p:
            del self.pesos_p[(i,j)]
         if ((j,i)) in self.pesos_p:
            del self.pesos_p[(j,i)]
   flujo, tiempo = self.ford_fulkerson()
   if flujo ==0:
      break
```

Para eliminar aristas se tomaron dos índices $i$ y $j$ de forma aleatoria dentro del rango de vértices existentes del grafo y si existía una arista entre estos se eliminaba, hasta eliminar un total de $p$ aristas.

```
while flujo!=0:
   a=0
   while a <p:
      (i,j)=random.sample(range(self.nn),2)
      if ((i,j)) in self.pesos:
         del self.pesos[(i,j)]
         del self.pesos[(j,i)]
```

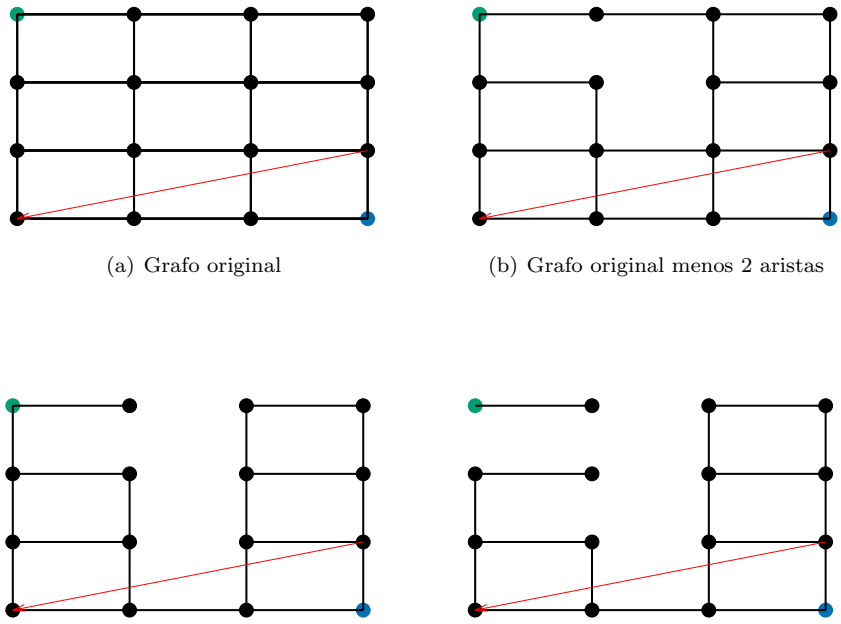
# 4. Resultados

Para mostrar cómo funciona la función de eliminar aristas, en la figura 1 se muestra un grafo de 4 por 4 nodos. En el inciso $a$ se presenta el grafo completo, conectado con una distancia máxima de Manhattan igual a uno, en el inciso $b$ se aprecia cómo es que son eliminadas 2 aristas, en el inciso $c$ se han eliminado 4 aristas y por último en el inciso $d$ se han eliminado 6 aristas y aquí se detiene la eliminación de aristas porque ya no existe algún camino que nos lleve desde el inicio (punto verde) hasta el final (punto azul).

Con el objetivo de observar el efecto de eliminar nodos y aristas de forma separada sobre un grafo conectado con distancias Manhattan y aristas aleatorias, sobre los tiempos para obtener el flujo máximo usando el algoritmo de Ford Fulkerson, se realizó un experimento de análisis estadístico. En este caso las capacidades de las aristas formadas por las distancias Manhattan provenían de una distribución normal con media de 6 y una desviación estándar de 1, y las capacidades de las aristas asignadas de forma aleatoria provenían de una distribución exponencial con un valor lamda de 0.1, la distancia de Manhattan

máximo fue $l = 2$ y el grafo está formado por 100 vértices en una malla de 10 por 10.

En la figura 4 se muestra un diagrama de cajas sobre el tiempo para obtener el flujo máximo al eliminar 10 aristas en cada iteración. Se puede observar como hay una tendencia a la baja a medida que eliminamos aristas, además se observaron caso en los que era necesario eliminar más de 900 aristas para poder eliminar todos los caminos de inicio a fin.



(a) Grafo original

(b) Grafo original menos 2 aristas

(c) Grafo original menos 4 aristas

(d) Grafo original menos 6 aristas

Figura 1: Eliminación de aristas.

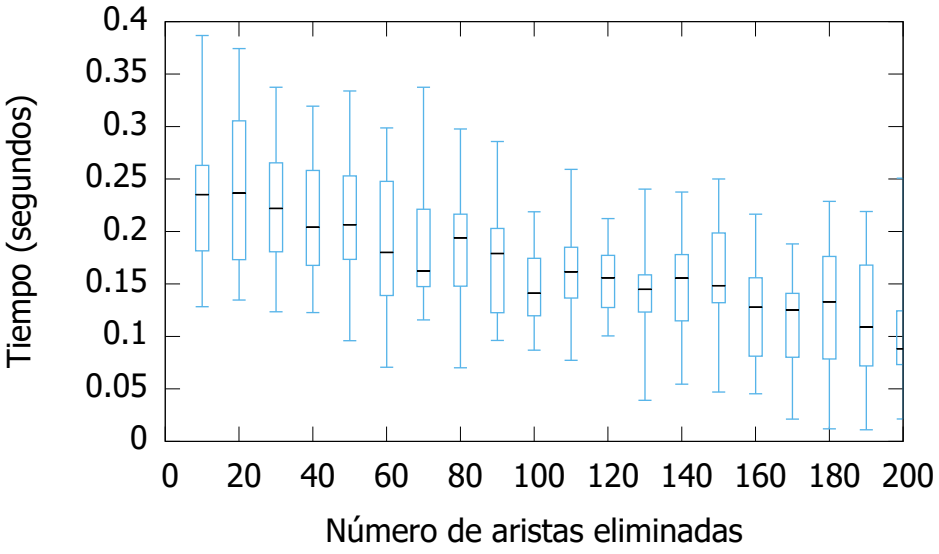



Figura 2: Grafo con $k$=100, $l = 2$ y eliminacion de 10 aristas en cada paso

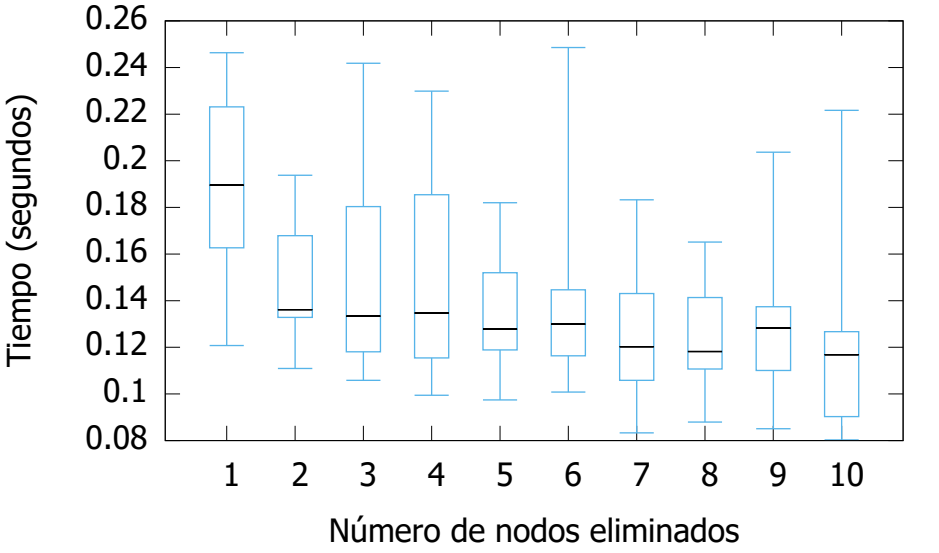



Figura 3: Grafo con $k$=100, $l = 2$ y eliminacion de 1 arista en cada paso.

En la figura **??** se muestra un diagrama de cajas sobre los tiempos para obtener el flujo máximo al eliminar 1 nodos en cada iteración. En ella se observa de nuevo como hay una tendencia a la baja a medida que eliminamos vértices y que en algunos caso fue necesario eliminar hasta casi 80 vértices para obtener un flujo de cero (no hay camino entre el inicio y el fin).

# 5. Conclusión

Como podemos observar en ambos casos eliminando nodos o aristas a medida que se vayan avanzando en las iteraciones, los tiempos para encontrar el flujo máximo disminuyen, pero en el caso de eliminar nodos se nota más el efecto sobre los tiempos de Ford Fulkeron, debido a que cada vez que se elimina un vértice eliminamos un bloque del grafo (vecindad), mientras que al eliminar aristas estas se pueden eliminar de diferentes regiones de grafo, dejando más opciones de búsqueda para el Ford Fulkerson.

# Referencias


[1] Alcantar G. Alan, *implementación de los algoritmos Ford-Fulkerson y Floyd-Warshall*, https://github.com/alan-arnoldo-alcantar/flujos/blob/master/Reporte

[2] Shirkhorshidi AS, Aghabozorgi S, Wah TY, (2015), *A Comparison Study on Similarity and Dissimilarity Measures in Clustering Continuous Data*, https://doi.org/10.1371/journal.pone.0144059

[3] *Mutzell, M., Josefsson, M.*, (2015), *Max Flow Algorithms?: Ford Fulkerson, Edmond Karp, Goldberg TarjanComparison in regards to practical running time on different types of randomized flow networks.*, http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-168027