Optimización de Flujo en Redes: Distancia Manhattan

Alcantar Gómez Alan Arnoldo
Matricula: 1935040

6 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))
```

```
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))
```

```
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
```

```
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
```

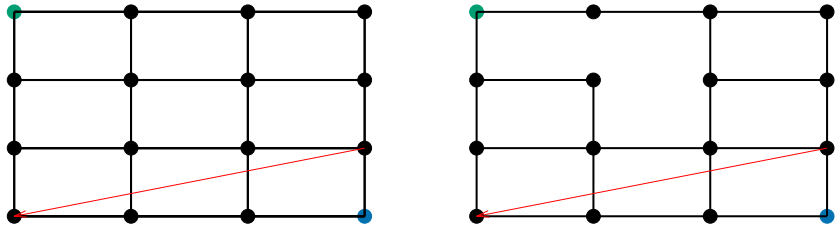
```
while flujo!=0:
```

```
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)]
```

---

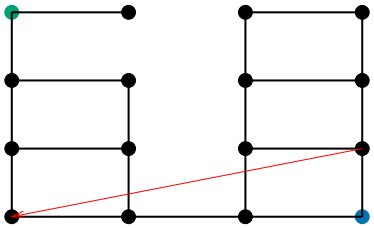
4. Resultados

A continuación, se presentan los resultados.

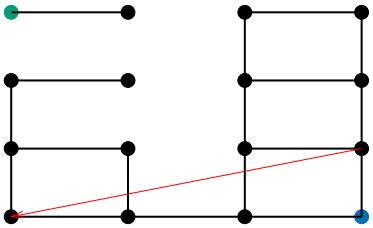


(a) Starks

(b) Arya y Reeds



(c) Lannisters



(d) Lannisters

Figura 1: Legos.

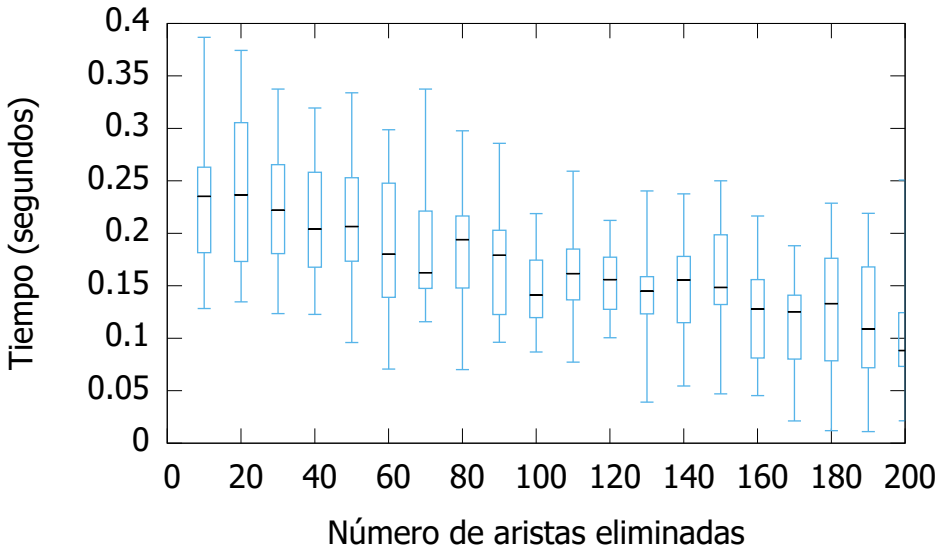


Figura 2: Gráfica de interacciones, $N = 10$, $T_{max} = 100$ y $\alpha = 0{,}7$, *modo* : *simple*.

5. Conclusión

Como se observó en la parte de experimentación el utilizar *clases* y *funciones* permite que podamos generar varios tipos de grafos a través de los parámetros con los que trabajen cada una de las *funciones*. Esto nos permite utilizar el mismo código para resolver distintos problemas donde las direcciones o capacidades entre los nodos tengas una característica importante.

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

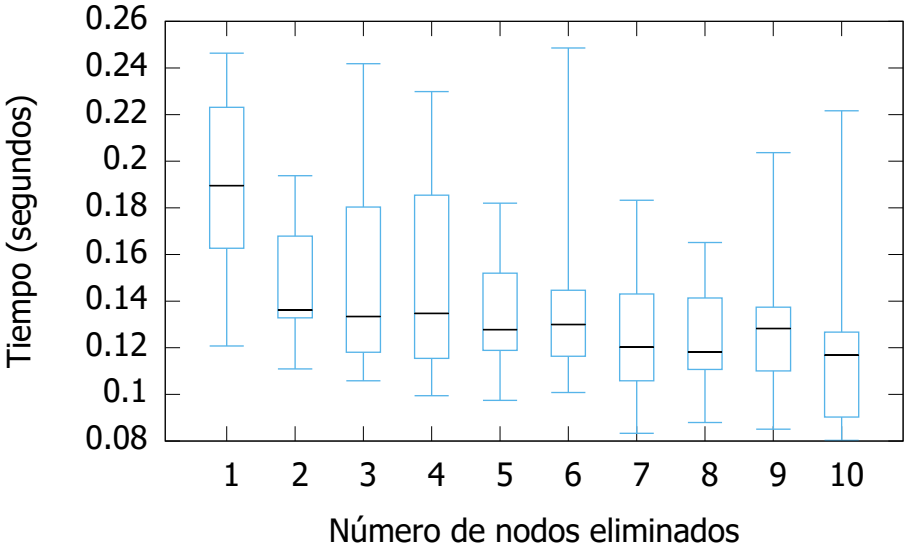


Figura 3: Gráfica de interacciones, $N = 10$, $T_{max} = 100$ y $\alpha = 0{,}7$, *modo* : *simple*.