Optimización de Flujo en Redes: Reporte 4

Mayra Cristina Berrones Reyes. 6291

Abril 2018

Introducción

Para este reporte se estarán utilizando tanto nuevos conceptos, como los que ya se habían implementado en reportes anteriores. Por ejemplo, en caso de nuevos conceptos, se estarán dando a conocer temas como clustering (también conocido como agrupamiento) es una de las técnicas de minería de datos el cual consiste en la división de los datos en grupos de objetos similares. Cuando se representan la información obtenida a través de clusters o racimos se pierden algunos detalles de los datos, pero a la vez se simplifica dicha información. [3]

En este caso, hablaremos de agrupamientos en relación con los grafos que se han estado realizando a lo largo del semestre.

El segundo concepto que estaremos visitando, es de la densidad de nuestro grafo. Esto se irá desarrollando a lo largo de este trabajo, basándonos en el número de aristas que conectan a los nodos, así como de algunos cálculos de promedios que se explicarán más adelante.

Modificación del grafo principal

Primeramente, iniciamos con una nueva forma del grafo. Ya no simplemente hacemos que los nodos se acomoden de manera aleatoria a lo ancho y largo del plano, si no que se les da un acomodo circular para poder apreciar mejor las conexiones que se tienen entre ellos y poder hacer un análisis más a fondo del porqué de estas conexiones.
La forma circular de los nodos se logró gracias al siguiente extracto de código, que se tomó del archivo *intento.py*.

```
def puntos (self, num, k):
   self.n = num
   self.k = k

   th = 2*(pi)/self.n
   with open ("grafica.dat", 'w') as salida:
      for i in range(self.n):
         x = sin(th*i)
         y = cos(th*i)
         self.P.append((x, y, i))
         self.nodos.append((x,y))
         if not (x, y) in self.vecinos:
            self.vecinos[(x,y)] = []
         print (x, y, i, file = salida)
```

A grandes rasgos, lo que queremos recalcar aquí, es el hecho de que se utilizaron las funciones de la librería de *math* de *Python* para poder utilizar las funciones del *seno* y *coseno*, lo que facilito mucho más poder posicionar los nodos en su forma circular.

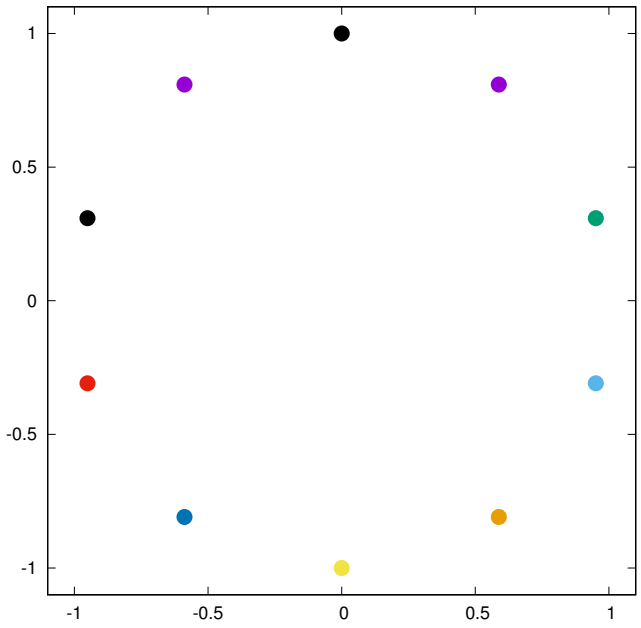

Figura 1: Nodos acomodados de manera circular dentro del plano.

Conexiones principales y secundarias

Antes de iniciar con los conceptos de agrupamiento y densidad que se mencionaban en la parte de introducción, para realizar las conexiones de los nodos se realizará una variación de lo que se hacía en reportes anteriores. Por ejemplo, en el reporte 3 [1]se tomaba una probabilidad al azar gracias a un número que se le asignaba con la herramienta *Random()* de *Python*, y dependiendo de este número aleatorio, era la cantidad de aristas que estarían asignadas a cada nodo. En dicho reporte, también manejábamos direcciones y pesos. Esto se deja un poco de lado en este trabajo. Lo que rescatamos de aquí un poco, es la forma de unir los nodos. Primeramente, tomaremos un valor K, el cual se tiene que asignar un valor más bajo que la mitad de los nodos totales por el siguiente motivo. El objetivo de esta variable K es juntar a un nodo con su k- ésimo vecino por ambos lados. Si tenemos por ejemplo, un grafo de 10 nodos y queremos darle una k de 6, esto sería redundante ya que ya existiría una arista que está uniendo a los últimos nodos que se quisieran emparejar.

```
def k_conect(self, k):
   for uno in range(1, k +1):
      for dos in range(0, self.n):
         u = self.nodos[dos]
         v = self.nodos[dos - uno]
```

```
self.aristas[(u, v)] = self.aristas[(v, u)] = uno
if not u in self.vecinos[v]:
   if u is not v:
      if v is not u:
         self.vecinos[v].append(u)
if not v in self.vecinos[u]:
   if v is not u:
      if u is not v:
         self.vecinos[u].append(v)
```

Como se mencionó, el peso y dirección no importan para este grafo, por lo que si conecto un nodo a, con un nodo b, entonces a es vecino de b, como b es vecino de a.

Una vez que entendemos este concepto podemos pasar a la siguiente forma de conexión. Se realizará una revisión para los nodos que no están conectados, y ahora si como se aplicaba en reportes anteriores [1], se hace uso de una variable aleatoria con la cual voy a decidir que nodos voy a unir si la variable k no logro unirlos.

Una vez que tenemos un grafo de esta manera, podemos continuar.

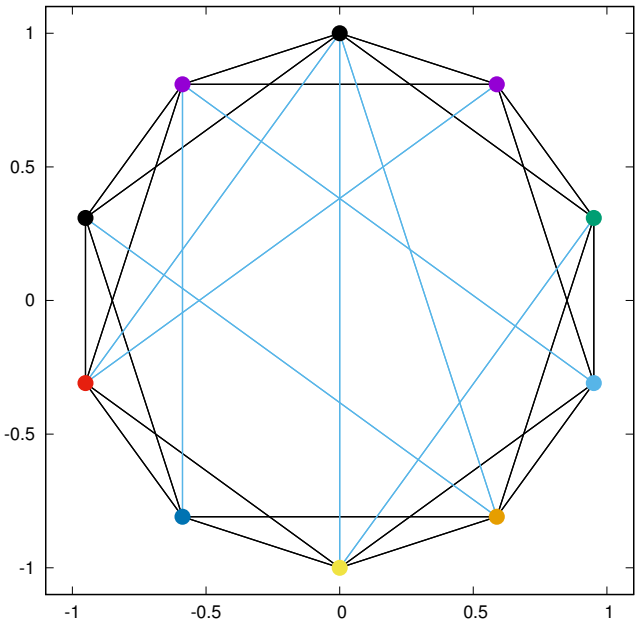

Figura 2: Grafo de 10 nodos. Su variable k esta puesta a 2 y su probabilidad esta en 0.2. Las lineas negras son las que representan la variable k, mientras que las azules representan la probabilidad aleatoria.

Densidad del grafo y agrupamiento

En este caso, se agregaron dos funciones al programa intento.py, que tiene funciones parecidas a los trabajos anteriores en cuanto a vecindades, vértices y aristas. Para poder realizar la parte de densidad, hacemos uso primero de la función que se utilizó en el Reporte 3, *floyd-warshall* [1] Estas líneas de código nos darán como resultado una formación de los nodos con las aristas que los unen, y de esta manera podemos sacar un promedio general de su densidad.

```
def argdist(self):
   self.sumaDist = 0
   for i in self.d:
      self.sumaDist += self.d[i]
   self.promDist2 = self.sumaDist/((self.n * (self.n - 1))/ 2)
   self.arg2 = self.promDist2 / (self.n - 1)
   with open("Prom_dis.csv", 'a') as salida:
      print(self.arg2, file = salida)
```

En este caso, lo que utilizamos del del algoritmo floyd-warshall es la información que nos proporciona el self.d. Con esto sumamos la distancia, y al final lo dividimos por una cota superior. En este caso, se eligió decir como cota superior el número total de nodos menos uno. La razón para este razonamiento es que, el peor de los casos es que los nodos se encuentren conectados solamente por una arista uno entre otro, por lo cual para poder recorrer todo el grafo tiene que pasar estrictamente por un solo camino hasta llegar al nodo inicial.

En el caso del agrupamiento es un tema un poco más complicado de explicar. Lo que se nos pedía era, tomar un nodo x, revisar cuales eran los vecinos con los que estaba conectado, y después hacer una revisión de si removíamos dicho nodo x, cuantas aristas quedaban conectadas entre los nodos vecinos (las aristas que tenia conectadas a x ya no cuentan). Después, hacer un promedio general de cada uno de los nodos, para saber el nivel de conectividad que existe entre los nodos.

```
def clustercoef (self):
   with open("Cluster_coef.csv", 'at') as salida:
      self.clus = []
      for (x, y) in self.nodos:
         self.cff = []
         idvec = len(self.vecinos[(x,y)])
         self.clustcoef2 = 0
         for t in range(0, idvec - 1):
            self.clustcoef = 0
            h = self.vecinos[(x,y)][t]
            if(x,y) is not h:
               for m in self.vecinos[h]:
                  if(x,y) is not m:
                     if m is not h:
```

```
                        if h is not m:
                           if h is not (x, y):
                              if m is not (x, y):
                                 if m in self.vecinos[(x,y)]:
                                    if (h,m) not in self.cff:
                                       self.cff.append((h, m))
                                    if (m, h) not in self.cff:
                                       self.cff.append((m, h))
                                       self.clustcoef +=1
            self.clustcoef2 += self.clustcoef
         dens = (self.clustcoef2) / (((idvec) * (idvec - 1)) / 2)
         self.clus.append(dens)
      c = 0
      for d in range(0, len(self.clus) - 1):
         c += self.clus[d]
      self.DensidadProm = c / len(self.clus)
      print(self.DensidadProm, file = salida)
```

Para poder dar revisión a los cambios que se presentaban dentro de los promedios tanto de distancia como de densidad, se iteraron varias veces las funciones, cambiando en cada cuantas la probabilidad de conexión para ver como afectaba esto a los resultados finales.

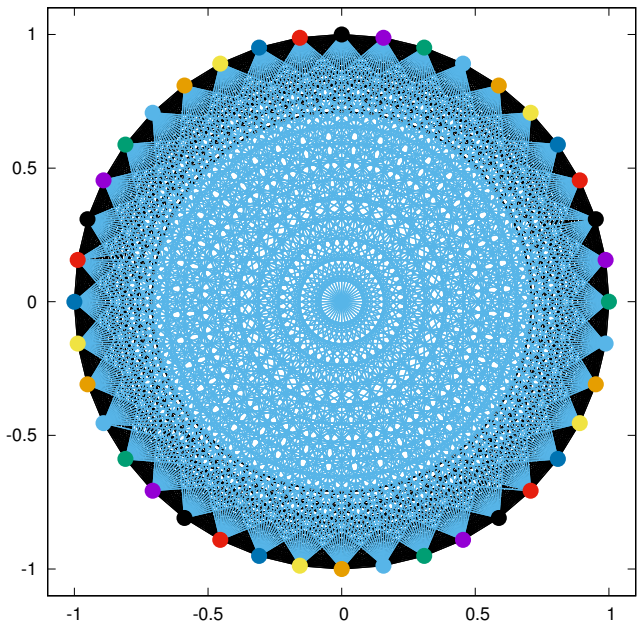

Figura 3: Grafo de 40 nodos con una k baja, de 5 pero una probabilidad aleatoria alta de 0.9.

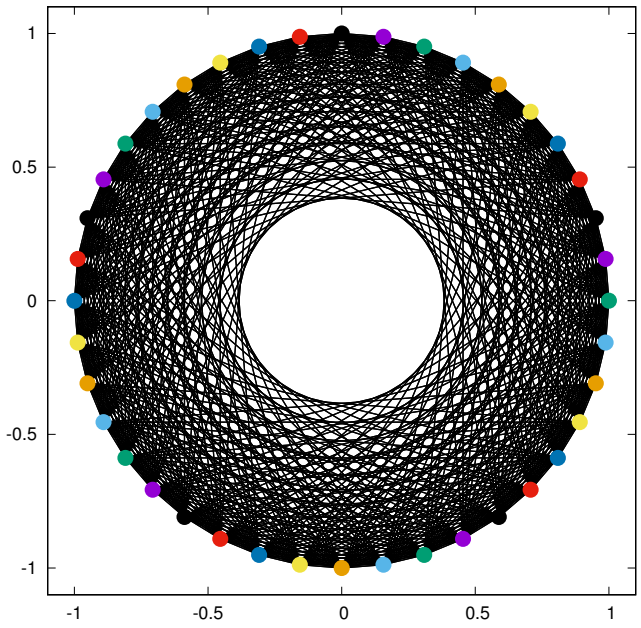

Figura 4: Grafo de 40 nodos con una probabilidad aleatoria baja, pero con una k alta de 10.

Resultados

Como resultado final se quería mostrar una gráfica en la cual se apreciará el comportamiento de dichas funciones con diferentes probabilidades. Para este caso se tuvo que realizar un extracto de la información que arrojaba el programa, ya que agrupaba todas los promedios y probabilidades en un solo archivo de texto. Al final, se muestra también la gráfica con todas las líneas que representan todas las demás corridas.

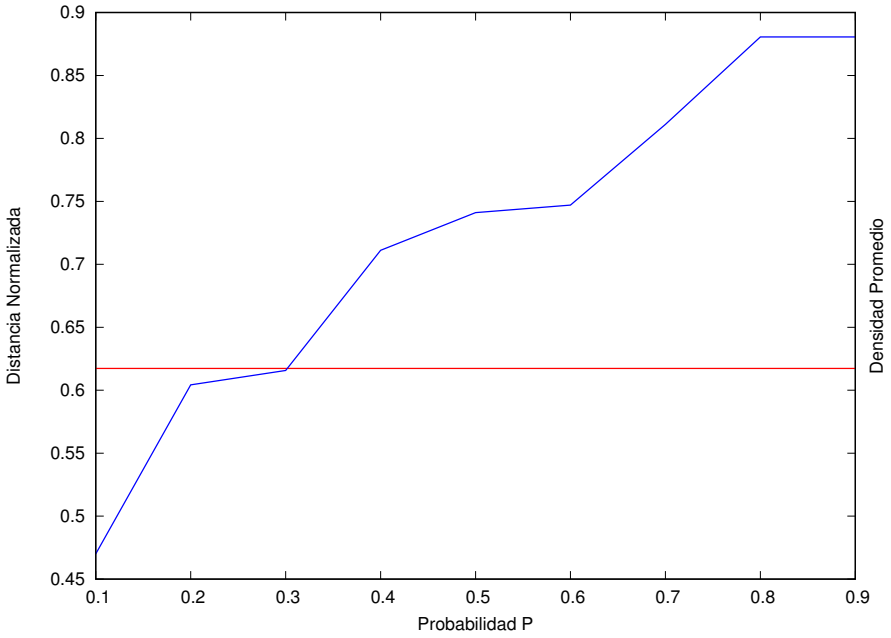

Figura 5: Gráfica de distancia y densidad de un extracto de las corridas. El grafo utilizado fue de 10 nodos, y la k se quedó en un valor bajo de 2. La linea azul representa el promedio de agrupamiento, mientas que el rojo es el promedio de distancia.

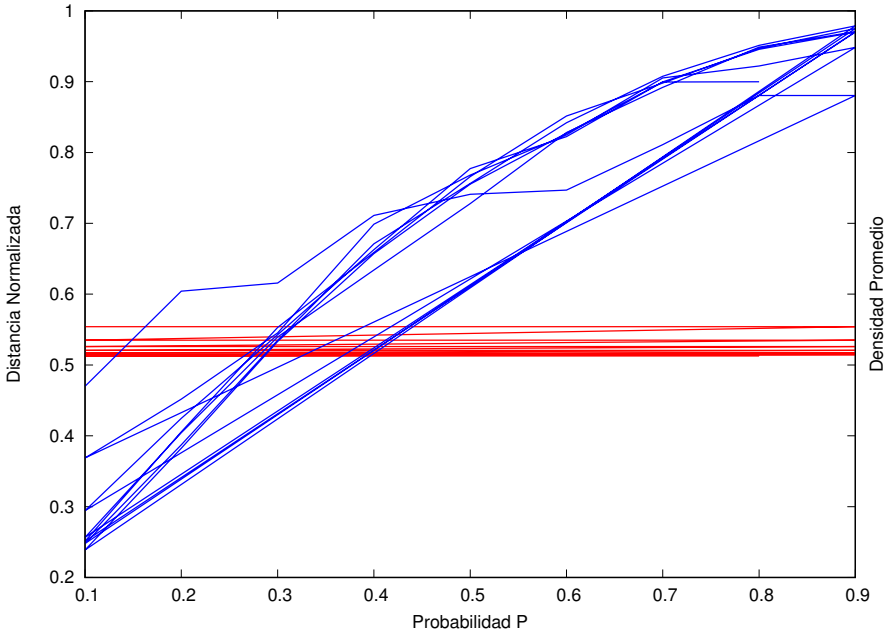

Figura 6: Gráfica de distancia y densidad de todas las corridas realizadas por el programa. La linea azul representa el promedio de agrupamiento, mientas que el rojo es el promedio de distancia.

Conclusiones

Este tipo de problemas y la manera en que se nos planteo, es una manera bastante interesante de darnos cuenta de que manera se puede utilizar el algoritmo de floyd-warshall, así como también ir tomando una idea de cómo funciona el agrupamiento, y los posibles resultados de tener que remover tal o cual nodo.

En cuanto a dificultad, la parte de tener que realizar los promedios de las distancias no resulta tan complicado una vez que tienes bien claro el concepto de cómo funcionaba el algoritmo de floyd-warshall, y saber aplicarlo correctamente.

En el tema del agrupamiento, fue un tema un poco mas complicado, ya que a primeras impresiones, y en los primeros intentos de los programas que se realizaron para este reporte, había una sensación de que se estaba quitando una pieza importante, y esto dejaba buscando a ciegas las partes que iban emparejadas con este, pero una vez que se experimentó con grafos pequeños, imprimiendo los vecinos para cerciorar cada paso, fue mas sencillo captar la idea.

Referencias

[1] Berrones Reyes, Mayra Cristina *Reporte3. Marzo 2018*

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

[3] Clustering *ECURed, https://www.ecured.cu/Clustering* **Consultado en Abril 2018**