Experimentos con el algoritmo de Floyd-Warshall en `python`

Yessica Reyna Fernández

Flujo en Redes

23 de abril de 2018

1. Introducción

En un grafo se pueden denotar la forma de relacionarse con otras personas, entonces si se formara un circulo con un determinado número de personas, todos pueden ver a todos, pero no cualquiera puede hablar o conversar con todos los que están en la circunferencia que se ha formado, ya que por la lejanía se genera ruido y no se puede transmitir de buena manera algún mensaje determinado, es por eso que de cierta manera solo se podría interactuar primeramente con el más próximo que es las dos personas a los lados y algunas más que son sucesivas a las próximas, dicho esto es una forma de interpretación a la nueva modelación de grafo que se genera y da explicación en las siguientes secciones y la forma de ver estas relaciones o conexiones en formas de aristas que perciben una distancia entre cada par de personas o nodos y que otras características más podrían ser validad para alguna relación que pueda ser interpretada por el grafo en particular.
Todo siguiendo la estructura para la realización de grafos prevista por la practica 3 [1].

2. Grafo circular base

Primeramente con respecto a la posición de los nodos, se posicionan en forma de circunferencia, entonces se realiza la transformación a coordenadas en radianes para la circunferencia a formar; a continuación se debe definir en ese caso la posición del ángulo en donde ira cada punto para esta transformación por lo cual se deberá tomar una medida equitativa para la cantidad de nodos que se tiene, tomando en cuenta la ecuación paramétrica de la circunferencia con centro en (a, b) donde $x = a + rCos(t)$ y $y = b + rSen(t)$ con $t \in [0, 2\pi]$, definiendo lo siguiente como parte básica del grafo:

```
1 angulo=2*pi/n #n=cantidad de nodos
2 r=0.3 #radio fijo de la circunferencia
3 c=(0.5,0.5) #posición del centro de la circunferencia
4 self.pos[v] = (c[0]+(r * cos(angulo * v)), c[1]+(r* sin(angulo * v))) #dentro de la
      función nodoscrear() con v=indice del nodo a crear
```

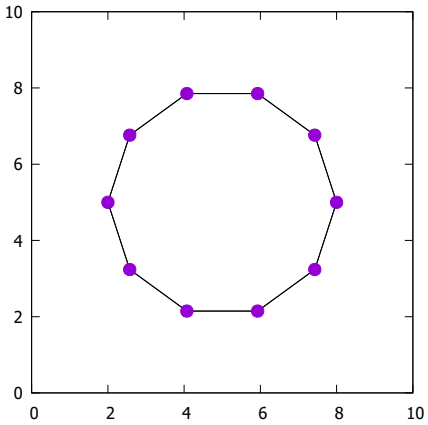
Ahora bien hablando de la forma de trazar las aristas entre nodos se mide en forma de un parámetro k, el cual varía dependiendo de la cantidad de nodos; por lo que se busca que cada vez que se corra el código las repeticiones que habrá de cada tamaño este dada por el valor $\lfloor n/2 \rfloor$ ademas de que el valor de k realiza las conexiones entre nodos; asimismo se define `E[(u,v)]` como las distancias euclidianas entre cada par de nodos y `vecinos[v].add(i+(j+1))` lo que agrega al nodo $i + (j + 1)$ entre los vecinos de v, todo esto descrito en lo siguiente:

```
def conecta(self, k) #función de conexiones en aristas
    for j in range(k): #recorrer tamaño de k
        for i in range(len(self.V)): #recorriendo cada nodo
            if i< (len(self.V)-(j+1)): #siempre que el indice no pase el valor k
                self.E[(i, i+(j+1))] = self.E[(i+(j+1), i)]=self.euclidiana(i,(i+(j+1)))
                self.vecinos[i].add(i+(j+1))
                self.vecinos[i+(j+1)].add(i)
```

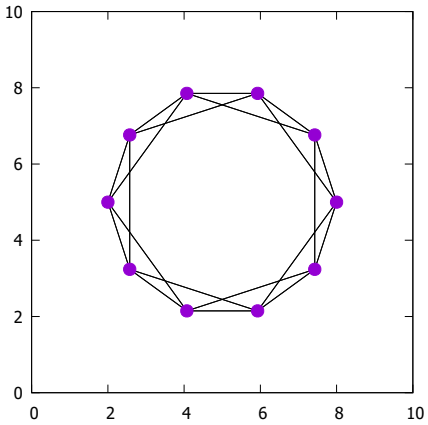
En relación con esto se define la función para integrar aristas aleatorias al grafo que depende de una probabilidad que aumenta con el valor de k, agregando esa nueva arista siempre que no se encuentre considerada en él, además de que su probabilidad para entrar al grafo sea más pequeña que un número que se genera aleatoriamente:

```
prob=2**-(k)
def conectaaleatorio(self, prob)
    if m is not w and (m,w) not in self.E:
        if random()< prob:
            self.E[(m,w)]=self.E[(w,m)]=self.euclidiana(m,w)
            self.vecinos[m].add(w)
            self.vecinos[w].add(m)
```

Así pues, esto definiría lo más básico por elaborar par aun grafo base circular, tratando en las siguientes secciones otros aspectos que se pueden agregar al grafo, a continuación se muestran unos ejemplos del grafo ya descrito:



(a) $n = 10$ y $k = 1$.



(b) $n = 10$ y $k = 2$.

Figura 1: Ejemplos de grafos circulares.

3. Promedios de distancias

Como se ha tratado en prácticas anteriores, el algoritmo de Floyd-Warshall [2] crea un vector con todos los pares de distancias entre un par de puntos dados u y v. Por lo cual se puede considerar como característica en el grafo lo que sería el promedio de distancias entre nodos, tomando la suma de distancias de todos los pares de nodos dados obteniendo así este parámetro que varía dependiendo de la cantidad de nodos, esto es, $D = \frac{\sum_{i=1}^{n} d[i]}{n}$, donde `d[i]` es el vector de las distancias tomado del algoritmo de Floyd-Warshall.
Donde todo lo ya dicho se desglosa en las siguiente líneas de código:

```
1 def promediodistancias(self):
2     self.suma = 0
3     for key, value in self.d.items():
4         self.suma= self.suma+value
5     return self.suma/n
```

Acto seguido a esto se define una nueva función matemática, la cual en relación con los valores de n y de k, se crea una cota superior al valor de la distancia promedio que se calculó con anterioridad.
Hablando un poco más de ella, esta se calcula en base a la longitud del vector de

distancias, el cual crece conforme al valor de n, entre el cociente de la mitad de las interacciones que hay de nodos y la longitud de la circunferencia.
Dicho de otra manera, se mostrara a continuación las líneas de código pertenecientes a la descripción de la formulación de la cota superior para la normalización de las distancia promedio del grafo conforme a la cantidad de nodos y el valor de interacción entre ellos k, asimismo nos indica si para algún valor de n y de k no se llega a cumplir el valor de la cota y se pueda realizar el ajuste de esta:

```
def cota(self):
    circumferencelength= 2*(pi)*r
    supremo=(((n**2)-n)/((0.5)*k))*circumferencelength
    if supremo> self.suma/n:
        return (self.suma/n)/supremo
    else:
        print(n, k)
        print("No es buena cota")
```

4. Densidad de cluster

Como otro rasgo a denotar para un grafo se puede considerar la densidad de cada uno de sus cluster o grupo su equivalente en español, el cual indica el número de conexiones que debería existir por grupo que forma cada uno de los nodos individualmente, es decir, quienes de sus vecinos está conectado entre ellos mismos.
Formulando esto matemáticamente se obtendría un coeficiente de densidad de grupo en un rango de $(0, 1)$ con la siguiente formula: $C = \frac{2m}{n(n-1)!}$.
Obteniendo esto en el código con las siguiente líneas, en donde al formar una doble iteración en la lista de los vecinos de v se evita poner el dos de la formula y dividiendo al final entre la cantidad de nodos generados:

```
def promclusters(self):
    csuma=0
    for v in range(len(self.V)):
        m=0
        for i in self.vecinos[v]:
            for b in self.vecinos[v]:
                if b in self.vecinos[v]:
                    m += 1
                    csuma += m/(n*(n-1))
    return csuma/len(self.V)
```

5. Resultados

El objetivo principal de la practica era verificar el comportamiento del tiempo que se toma el algoritmo implementado en resolverse y comprobar así que se sigue manteniendo el mismo comportamiento de tiempo computacional empleado por el algoritmo de Floyd-Warshall, asimismo como interpretar los datos obtenidos por las nuevas características del grafo agregadas para uso de esta práctica y así demostrar algún tipo de comportamiento que se relacione entre la cantidad de nodos a considerar y las variaciones de conexiones entre pares de nodos.
Entre las siguientes figuras se mostrara la relación de los tiempos tomados para los

diferentes cantidades de nodos que se verá en la figura 2, hay que destacar que al analizar la gráfica generada por los tiempos esta sigue manteniendo el mismo tiempo computacional referido del algoritmo mismo de $\mathcal{O}(n^3)$.
Acto seguido se apreciaran los gráficos generados para las distancias promedio y las densidades de cluster realizados para cada cantidad de nodos graficadas contra la probabilidad con la que se varió las conexiones intermedias de nodos en una escala logarítmica en un intervalo de $(0, \frac{1}{2^1})$ en la figura 3 y la figura 4.

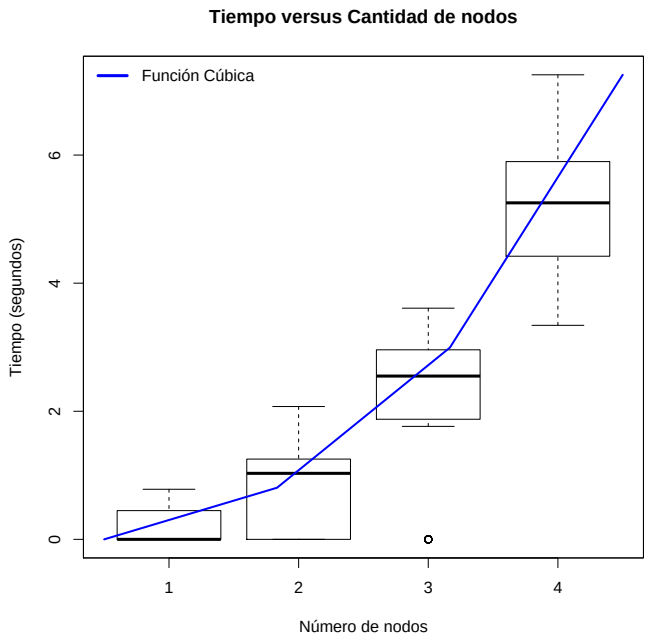
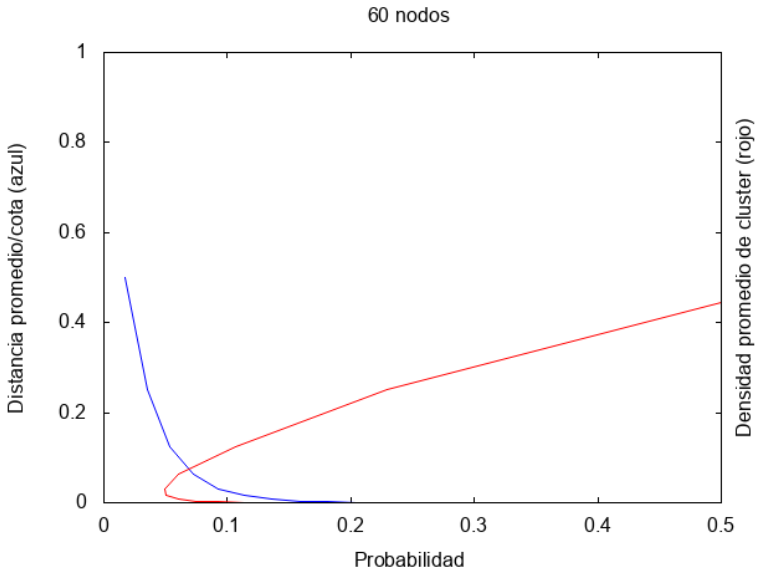


Figura 2: Tiempo versus Cantidad de nodos

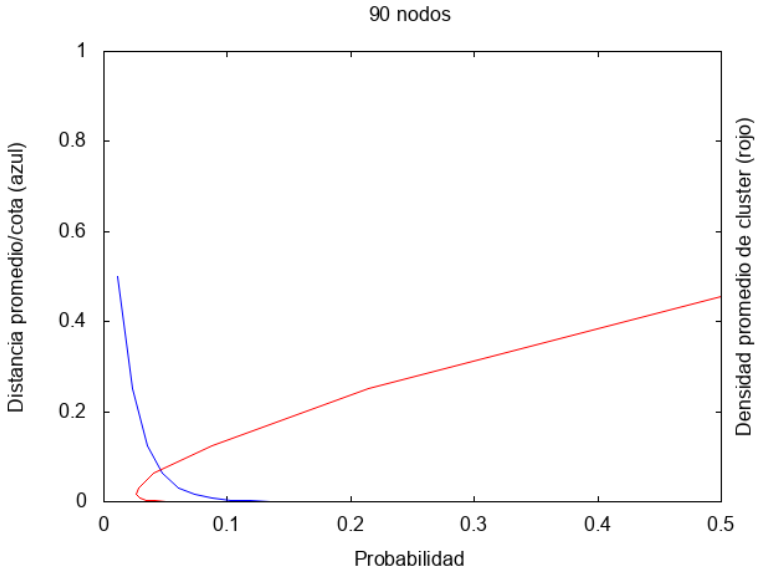
Para la gráfica anterior se hace uso del lenguaje de programación `R` [3] con apoyo de los archivos tipo `csv` con los datos de los tiempos para cada cantidad diferente de nodos, verificando la normalidad de los datos y ajustando la curva de esfuerzo computacional del algoritmo a los datos analizados en un diagrama de cajas, todo esto se resume con las siguientes líneas de código:

```
Tiempos <- as.data.frame(t(read.csv("Tiempo60.csv", header=FALSE)))
T90<-as.data.frame(t(read.csv("Tiempo90.csv", header=FALSE)))
T120<-as.data.frame(t(read.csv("Tiempo1.csv", header=FALSE)))
T150<-as.data.frame(t(read.csv("Tiempo150.csv", header=FALSE)))
TTotales<-t(smartbind(Tiempos, T90, T120, T150,fill=0))
shapiro.test(TTotales)#no normales
pdf("TiempovsN.pdf")
boxplot(TTotales, main="Tiempo versus Cantidad de nodos", xlab=c("Número
    de nodos"), ylab=c("Tiempo (segundos)"), varwidht=TRUE, notch=FALSE,
    plot=TRUE)
par(new=TRUE, pty="m", xaxt="n", yaxt="n")
plot(seq(1:4)^3, type="l" , xlab="", ylab="", col="blue", lwd=2)
legend("topleft",col=c("Blue"), legend =c("Función Cúbica"), lwd=3, bty =
    "n")
graphics.off()
```

Mientras tanto que las siguientes gráficas fueron realizadas en `gnuplot` apartir del archivo dentro del repositorio llamado `grafica.py` [4]

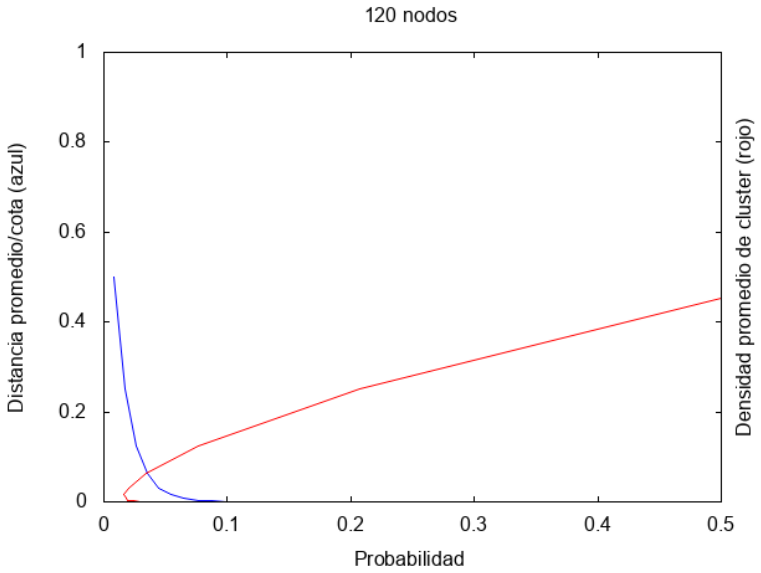


(a) $n = 60$.

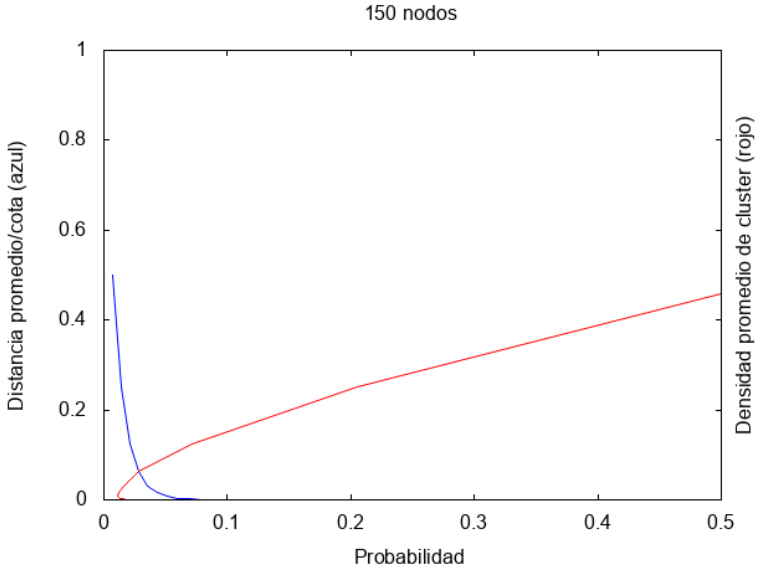


(b) $n = 90$.

Figura 3: Distancia promedio/cota y densidad de cluster.



(a) $n = 120$.



(b) $n = 150$.

Figura 4: Distancia promedio/cota y densidad de cluster.

En conclusión se puede asegurar que con forme la cantidad probabilidad de conexiones aumenta, es decir, hay más aristas en el grafo respecto a la distancia promedio generada por todos los pares de distancias toma valores cada vez más pequeños ya que la distancia decrece a grandes rasgos ya que hay más diferentes tipos de aristas por considerar, para la creación de distancias entre pares de nodos, lo cual las acerca más unos entre otros.
En lo cual el caso de la densidad de cluster o agrupación es un caso contrario, ya que entre más conexiones haya por nodo, habrá más posibilidad de que los vecinos del mismo tengan aristas entre ellos o la diferencia entre la cantidad de aristas que deba haber entre ellas a las que en realidad exista sea cada vez más pequeña.

Referencias

[1] Yessica Reyna Fernández. Tarea 3. 2018. https://github.com/YessicaFer/FlujoenRedes/blob/Tarea-1/Tarea%203/Tarea_3_Yessica.pdf

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest y Clifford Stein *Introduction to Algorithms*, capitulo 25. 2009.

[3] The R Project for Statistical Computing. https://www.r-project.org/

[4] Yessica Reyna Fernández. Tarea 4. 2018. https://github.com/YessicaFer/FlujoenRedes/blob/Tarea-1/Tarea%204/grafica.py