Medición experimental de la complejidad asintótica con `python` y `gnuplot`

Yessica Reyna Fernández

Flujo en Redes

7 de abril de 2018

1. Introducción

Un *grafo* es una representación simbólica de los elementos constituidos de un sistema o conjunto, mediante esquemas gráficos. Por lo cual se ha considerado a los grupos sociales como los elementos en particular de esta representación, buscando las relaciones ya sean de características entre ellos o que tan relacionados están unos de otros.
Así definiendo las conexiones que hay entre un par de nodos y las orientaciones o direccion de la arista entre cada par de ellos y por consiguiente, relacionar mediante una forma aleatoria una posible nueva amistad, o relación de algún tipo entre grupos o clases diferentes para generar conexidad entre los mismos.
Viendo así en la practica las relaciones ya establecidas anteriormente[1], pero con la introducción de dos nuevos procesos dentro del grafo.

1. Algoritmo de Floyd-Warshall[2]
2. Algoritmo de Ford-Fulkerson[3]

Los cuales serán descritos mas a detalle en la sección 3 y 4.

2. Grafo Base

El grafo base empleado durante la practica elabora grafos de tamaño de nodos especificados mediante la variable n, el cual elabora grafos simples, grafos ponderados, grafos dirigidos y grafos ponderados y dirigidos.
Para determinar si el grafo es ponderado o dirigido respectivamente se hace el uso de dos variables:

```
1 cap = False #Si cap=true el grafo sera con ponderacion en las aristas / false para no
      agregar ponderacion
2 orientado = True #Si orientado=true el grafo sera con orientacion en las
      aristas / false para no agregar orientacion
```

En donde el usuario podrá hacer control mediante estas variables sobre el tipo de grafo a crear.

Ademas para mejora del grafo y tener conectividad entre tipos de grupos de nodos, se agrego una nueva función, la cual agrega una nueva arista si el par de nodos seleccionados al azar de los ya existentes son diferentes, y siempre que también esa arista que se agregue no exista dentro del grafo, teniendo a lo más n nuevas aristas agregadas al grafo.

```
def yoconecto(self,n):
    q=randrange(0, n)
    p=randrange(0, n)
    f=randrange(0, 10)
    peso=choice ([1,2,3,4,5])
    if q!=p and (q, p, f, peso) not in self.aristas:
```

En donde q representa al primer nodo para la creación de la arista, p es el otro nodo con el cual conecta q, y se le da el mismo formato que al resto de las aristas ya existentes en el grafo; f representa el color de la arista a agregar y *peso* indica el tipo de linea a usar para representar la capacidad de la arista.
Agregando como adicional al código el cambio de las ponderaciones para las aristas, las cuales dependen de la variable *peso* para tomar un rango de ponderación. Presentando a continuación un ejemplo para las variaciones descritas:

```
if peso==1:
    tamaño=randint(1,5)
    self.pesos[(x1,y1),(x2,y2)]=tamaño
```

Haciendo la variación para cada uno de los tipos de lineas restantes con un incremento de 5 a la vez para cada uno de ellos. Algunos ejemplos de estas figuras se encuentran en la figura 4.

3. Algoritmo Floyd-Warshall

El *algoritmo de Floyd-Warshall* resuelve el problema de todos los pares de caminos más cortos en un grafo dirigido. El cual tiene un orden computacional de $O(V^3)$.
Entonces habiendo definido de forma general el funcionamiento del algoritmo, en otras palabras, lo que busca resolver este algoritmo, se agrega este mismo al código de grafo base, adaptando el código revisado en clase[4] y regresando como resultado el conjunto de las distancias entre los nodos, para el cual, ente caso se ha definido como la ponderación de las aristas, guardando el resultado de todas las instancias en las siguientes lineas:

```
with open("Floyd-Warshall.dat", "at") as archivo:
    print(d, file =archivo)
```

4. Algoritmo Ford-Fulkerson

El *método de Ford-Fulkerson* resuelve el problema de flujo máximo, el cual usa tres ideas muy importantes: redes residuales, caminos de aumento y cortes. El cual tiene un tiempo computacional lineal $O(E|f^*|)$ donde f^* es el máximo flujo de la red.
Este método incrementa el valor del flujo iterativamente, iniciando con un flujo de

cero, siendo que este flujo incrementa al encontrar un *camino de aumento* el cual definimos como un camino simple del nodo inicio al sumidero en la red residual. En donde si se tiene una red de flujo G y un flujo f, la *red residual* consiste en las aristas que representan como se puede cambiar el flujo en cada una de las aristas de G; ademas de contener aristas que no están en G.
Teniendo como ultimo elemento los *cortes*, los cuales en una red de flujo es una partición del conjunto de nodos del grafo original dado en el nodo origen.

```
G.ford_fulkerson(1,n-1)
with open("Ford-Fulkerson.dat", "at") as archivo:
    print(maximo, file=archivo)
```

5. Medición experimental

Para la verificación del tiempo computacional de ejecución del código y creación de grafo, así como también la generación de los archivos de resultados, se agregaron las siguientes lineas al código:

```
if orientado is True:
    with open("tDirigido.csv","at") as hile:
        start_time = time.clock()
        print (time.clock() - start_time, file =hile)
```

Así también para la creación de las instancias y poder verificar en la siguiente sección la normalidad o no de los datos obtenidos de los tiempos se agregaron dos bucles, uno para la generación de los tamaños diferentes del grafo y otro para la repetición del tamaño de grafo, lo cual se podrá ver en las siguientes lineas del código.

```
for i in range (1,12):
    for b in range (0,6):
        n= i*10 #Cantidad de nodos para cada instancia
```

Se generaron cuatro tipos diferentes de archivos, uno para cada diferente tipo de grafo ya especificado con anterioridad, los cuales se guardaron en su defecto en un archivo `csv` identificado por el tipo de grafo generado, ademas de tener un archivo tipo `dat` para los resultados del algoritmo Floyd-Warshall y del Ford-Fulkerson.

6. Análisis Estadístico

Para la comprobación de los datos obtenidos de la medición experimental se genero un archivo tipo `R`, en el cual se exporta la base de datos del `csv` y se modifica para su mejor manejo de datos en un bucle.
Primeramente se busca verificar si los datos obtenidos de la experimentación son distribuidos normalmente para una mayor facilidad de análisis, mediante el uso de *la prueba Shapiro-Wilk* podemos verificar esta información, ya que es usada para contrastar la normalidad de los datos dados.

```
Dirigido <- read.csv("tDirigidoTT.csv", header=FALSE)
#El número 11 en los datos corresponde a la cantidad de tamaños diferentes de
    grafos y el 6 a las repeticiones de cada tamaño
TDirigido <- data.frame()
for (i in 1:11){
```

```
        TDirigido <- rbind(TDirigido, Dirigido[((6*i)-5):(6*i),])
}
shapiro.test(TDirigido[,1])
```

En donde el *valor p* es la probabilidad de que los datos se ajusten de mejor manera a una distribución normal. Entonces si p es menor que el valor de *alpha*, los datos obtenidos no se encuentran normalizados, por defecto la prueba se realiza bajo un valor de *alpha* de 0.05.
Después de verificar la normalidad o no de los datos se elabora algún tipo de gráfico con el cual se pueda saber mas acerca de ellos y verificar si siguen algún patrón lineal, cuadrático, cubico, exponencial, etc.
En el caso ideal los datos muestreados tendrán un comportamiento similar al orden computacional de los algoritmos introducidos en el código.
Mediante el uso de *diagramas de cajas* el cual esta basado en cuartiles y la visualización de la distribución de los datos, mostrando la información de los valores mínimos y máximos, y el uso del primer cuartil, la mediana y el tercer cuartil, así también muestra datos atípicos de los mismos y su simetría de distribución. En donde los *cuartiles* son valores que dividen una muestra de datos en cuatro partes iguales.
Ademas este tipo de gráfico contiene lo que denomina como *bigotes* que son las lineas que se extienden desde la caja formada por los datos ya mencionados hasta el valor máximo y mínimo o hasta 1.5 veces del valor de rango intercuartilico.

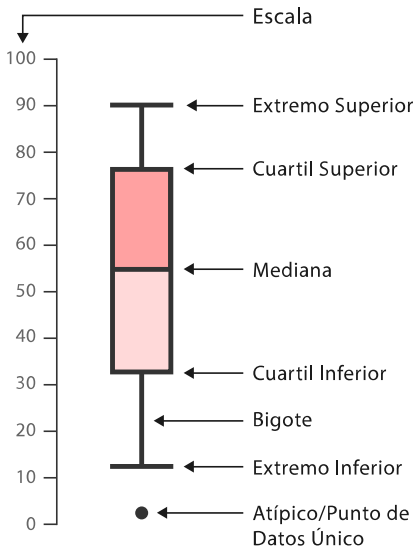


Figura 1: Ejemplo diagrama de caja.

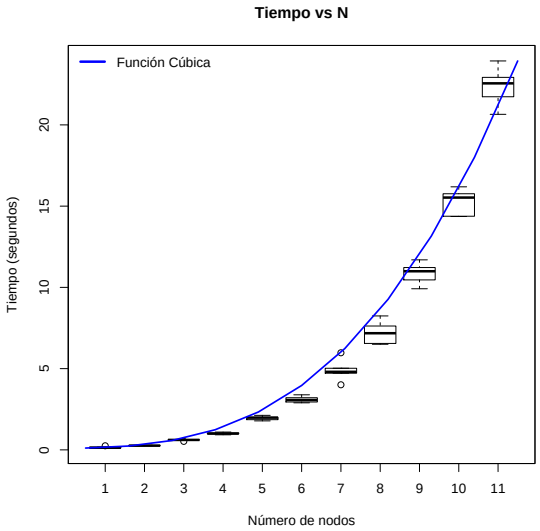
El paso próximo después de tener los conocimientos anteriores sobre como funcionan los diagramas de caja es iniciar el controlador gráfico del dispositivo para crear y exportar al formato que se desee la imagen del gráfico, en este caso se usara el formato `pdf`, ademas se hace uso de la función y los argumentos deseados para generar el diagrama de caja.
Como los datos generados de los tiempos fueron ordenados en filas, se necesita hacer una modificación más a los mismos para obtener una caja por cada tamaño de grafo.

```
pdf("TNoDirigidoTF.pdf")
boxplot(t(TNoDirigido), main="Tiempo vs N", xlab=c("Número de nodos"),
    ylab=c("Tiempo (segundos)"), plot=TRUE)
```

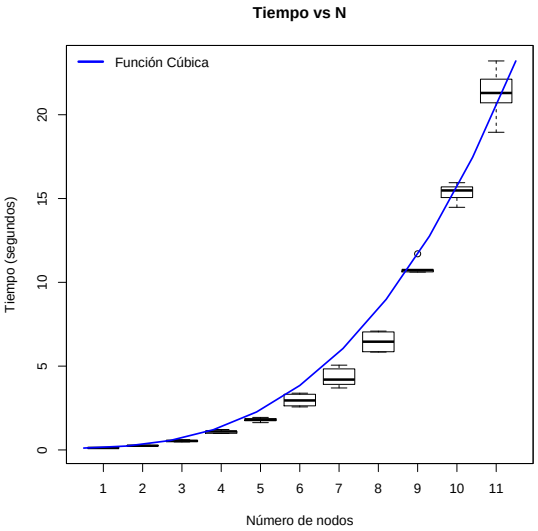
En donde `t` funciona para generar la matriz transpuesta de los datos usados o especificados como argumento de la función; ademas de que `main` indica el titulo del gráfico a elaborar, `xlab` y `ylab` configuran la etiqueta para los ejes del gráfico, y por ultimo `plot` hace referencia a mostrar el gráfico si es `TRUE` y en su caso contrario con `FALSE` regresa como dato el resumen de los datos usados para el diagrama de caja.

```
par(new=TRUE, pty="m", xaxt="n", yaxt="n")
plot(seq(1:11)^2, type="l" , xlab="", ylab="", col="red", lwd=2)
legend("topleft",col=c("red"), legend =c("Función Cuadratica"), lwd=3, bty =
    "n")
graphics.off()
```

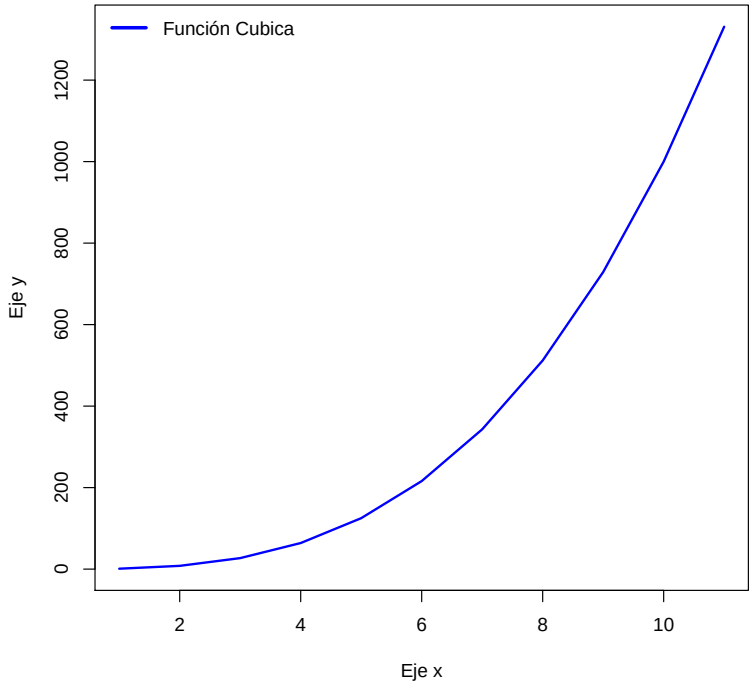
Haciendo uso de la función `par` la cual se esta empleando para establecer algunos parámetros de la función `plot` como seria `new`, para sobrescribir el gráfico descrito en `plot`, ademas de usar los parámetros `xaxt` y `yaxt` para especificar el tipo de etiqueta para los ejes del gráfico, en donde al usar n se suprime la visualización de la etiqueta deseada.
Ademas con `plot` especificado como función es usado para la visualización de un gráfico a partir de datos dados. En donde el primer parámetro fue especificado para el ajuste de la curvatura de los datos, y consecuentemente darle color y leyenda de la misma, ademas de como se busca que no se grafique nada mas que la curva y lo que es su leyenda, después usando el parámetro `bty` para suprimir la caja de la leyenda en la función `legend`.
A continuación se presentaran los resultados obtenidos para los cuatro archivos generados para cada diferente tipo de grafo comparados con la estructura teórica de los algoritmo implementados al código combinado sus tiempos computacionales en la figura 2.
Con los gráficos dados y los cálculos realizados en el `script` de `R` se puede verificar que la curva teórica de los algoritmos en conjunto, el algoritmo de Floyd-Warshall y Ford- Fulkerson, que como se menciono anteriormente cada uno de ellos tiene orden cubico y orden lineal, al combinarlos se sigue manteniendo la asintota de la curva en un orden cubico, en donde al contrastarla con los datos obtenidos del diagrama de caja de cada uno de los tipos de grafo es claramente una buena curva ajustada a los datos y por consiguiente se puede concluir que el código elaborado y usado para la experimentación mantiene el orden computacional de los algoritmos implementados ya mencionados con anterioridad para la elaboración de grafos, es decir es una buena inplementanción adaptada en especifico para este código en particular.



(a) Diagrama de caja de tiempos en un grafo dirigido y ponderado.

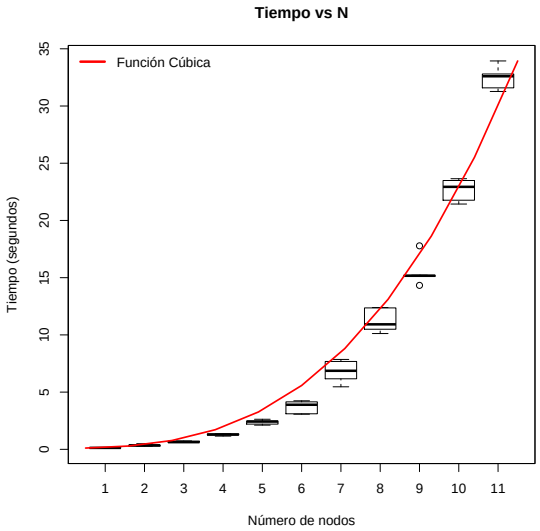


(b) Diagrama de caja de tiempos en un grafo dirigido.

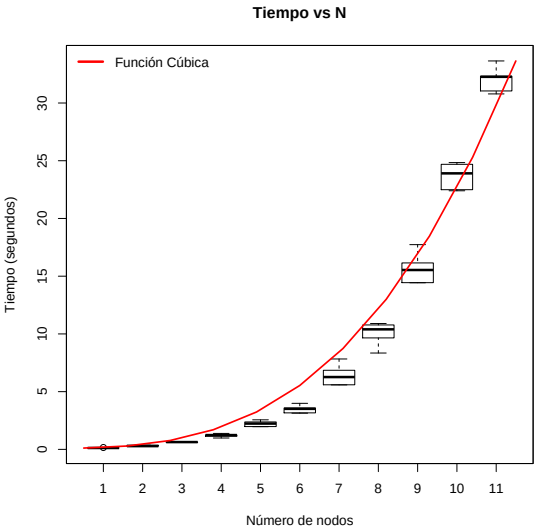


(c) Curva Teorica $O(n^3)$.

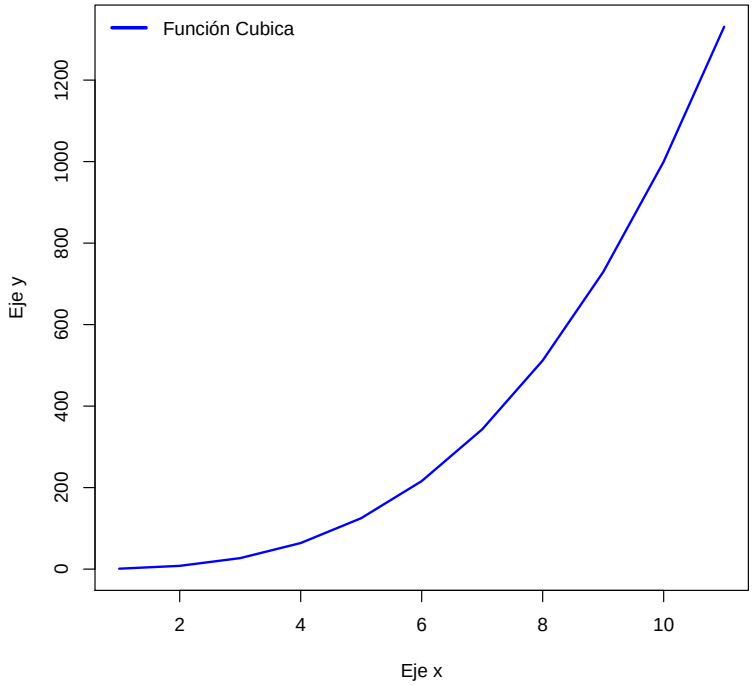
Figura 2: Tiempos de grafo dirigido vs Curva Teórica



(a) Diagrama de caja de tiempos en un grafo ponderado.

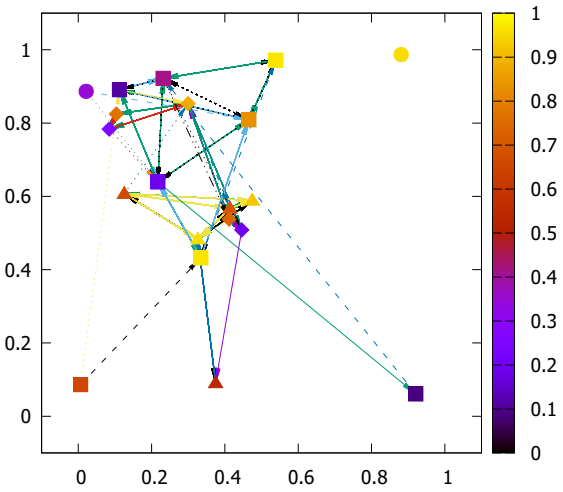


(b) Diagrama de caja de tiempos en un grafo simple.

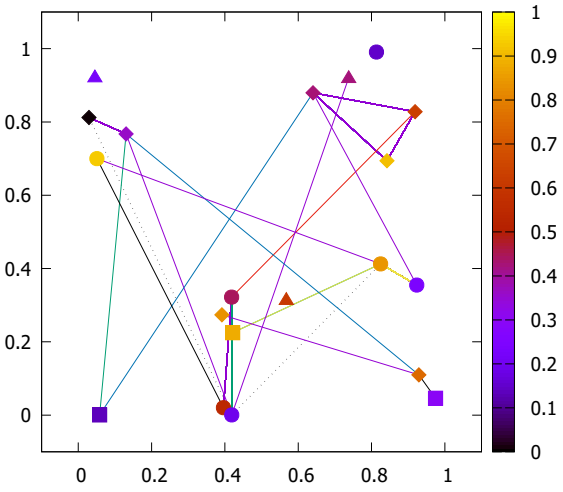


(c) Curva Teorica $O(n^3)$.

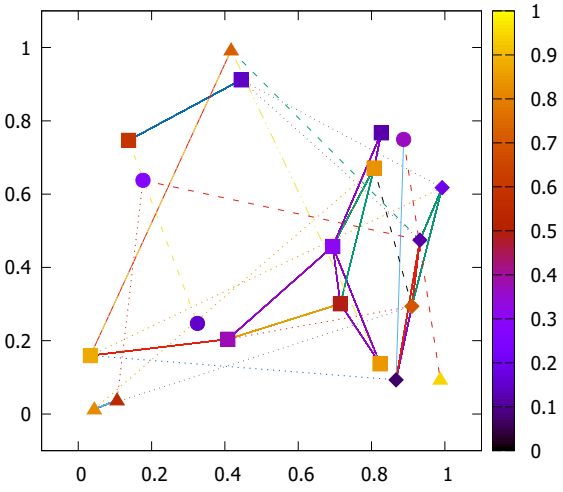
Figura 3: Tiempos de grafo no dirigido vs Curva Teórica



(a) Grafo dirigido y ponderado.



(b) Grafo simple.



(c) Grafo ponderado.

Figura 4: Ejemplos $n = 20$