REDES ALEATORIAS

Por Jorge Manrique y Javier Martín

Resumen

Estudio de las principales características y propiedades de los grafos, en espacial, de los grafos aleatorios. Búsqueda de aplicaciones para la vida diaria.

Introducción

Las redes aleatorias han sido un tema de estudio que proviene de la Teoría de Grafos. Ésta comenzó su desarrollo en el siglo XVIII con la resolución del problema de Los Puentes de Königsberg. Consistía en hallar un recorrido que recorriese la ciudad de Königsberg pasando por los 7 puentes de la ciudad solo una vez y volviendo al punto de origen. Desde entonces, los grafos han sido ampliamente estudiados y desarrollados. Una de las ramas más importantes sería la Teoría de Redes, que se encarga de la revisión de la estructura de las conexiones que tienen los grafos.

En este proyecto haremos una pequeña introducción a todos los conceptos básicos ligados a esta materia y ciertas figuras matemáticas en las que se basan. Reflexionaremos sobre cómo emplear de forma eficiente la aplicación Matlab para el estudio de las características generales de los grafos e indagaremos en los comandos que contiene la herramienta, además de ciertos usos prácticos que puedan llegar a tener.

Conceptos generales

Definiremos los conceptos más importantes de la Teoría de Redes, sobre los que versará el discurso del documento:

-Grafo: es un objeto matemático formado por nodos que representan relaciones binarias entre estos puntos mediante arcos.

-Vecinos: son puntos directamente conectados por un solo arco.

-Clúster: un conjunto de nodos conectados entre sí, de tal manera que se puede llegar desde un punto X a un punto Y pasando por otros nodos (Z, W, T, …) que también pertenezcan al mismo clúster.

-Red aleatoria: grafos de N vértices a los cuales se les asigna una probabilidad P de vincularse con cada uno de los demás nodos. En caso de que la probabilidad sea P=1 el sistema estará completamente conectado de manera que cada punto es vecino de todos los demás (un sistema de un solo clúster). Si la probabilidad es P=0, tendremos un sistema con N puntos aislados (un sistema de N clústeres).

Teniendo claros estos conceptos básicos podemos comenzar a estudiar diversas propiedades de los grafos o redes aleatorias. Aunque los grafos pudieran parecer un sistema bastante simple, podemos encontrar ciertas propiedades muy interesantes para su estudio estadístico.

Teoría

¿Cómo crear y representar redes aleatorias en Matlab?

En Matlab los grafos están determinados por una matriz de adyacencia o por un conjunto de dos vectores.

Una matriz de adyacencia es una matriz simétrica y cuadrada de dimensión igual al número de nodos, cuyos valores generalmente serán 0 y 1. El significado que hay detrás de esta matriz sería que los valores de los componentes de la matriz nos indican si 2 puntos están conectados. El valor del componente a_{12} nos indica si hay una conexión entre el punto 1 y el 2 (1 significa conectados y 0 desconectados). En grafos donde incluimos la distancia entre nodos, concepto que definiremos más adelante, los componentes podrán tener los valores positivos que se quieran (0

significará no conectados y cualquier otro número significará conectados a X distancia).

Los vectores en cambio relacionan los nodos conectados a través de las entradas del vector. Por ejemplo, los vectores $s = [1\ 1\ 2]$ y $t = [2\ 3\ 3]$, nos indican que el nodo 1 está conectado con los nodos 2 y 3 y que el nodo 2 está conectado con el nodo 3. Cabe destacar que el primer vector no contiene el nodo N mientras que el segundo vector no contiene al nodo 1.

Para la creación de un grafo aleatorio le damos una probabilidad P de unión entre 2 puntos, generamos un numero aleatorio entre 0 y 1 para las componentes de la matriz. En caso de que el numero aleatorio sea menor que la probabilidad, a cada componente de la matriz se le otorga el valor 1 (Si la probabilidad es 1, todos los números generados se convierten en uno, por lo que todo el grafo está conectado, si es 0 ocurre lo contrario). Luego utilizamos esta matriz para crear la matriz simétrica de adyacencia. Triangulamos superiormente la matriz y convertimos la diagonal a 0. Con esto podemos utilizar el comando "graph" para representarlo·

```
B=rand(n)<p;
for i=1:n
    B(i,i)=0.0;
end
B=triu(B);
B=B;
G=graph(B,'upper');
```

Código de creación de un grafo

La segunda forma de generar un grafo (con vectores), sólo la usaremos en las funciones sobre viajes, ya que es más complicado generar grafos aleatorios con este método.

Matriz de grado D

Con estos datos también podemos obtener una matriz de gran importancia: la matriz de grado D. Es una matriz diagonal que nos indica el número de vecinos de cada nodo, siendo la componente a_{ii} el número de vecinos del nodo i. Para su cálculo, se suman los valores de las columnas y se introducen dichos valores en la diagonal principal.

Matriz Laplaciana

Utilizando estas 2 matrices se puede obtener la matriz Laplaciana, una matriz muy útil para la teoría espectral de grafos. La fórmula de esta matriz es muy sencilla. Se obtiene restando a la matriz de grado la matriz de adyacencia.

$$L = A - B$$

Lo importante de esta matriz son sus autovalores. Los autovalores y autovectores de esta matriz nos dan mucha información sobre el grafo que representa. Por ejemplo, la multiplicidad del autovalor 0 nos indica el número de clústeres o subgrafos que tenemos. El segundo autovector no nulo nos describe la conectividad algebraica. Cuando el grafo está muy dividido este valor es cercano a cero y aumenta cuando los puntos están más conectados.

Distancias en un grafo

Las distancias en un grafo se definen como el mínimo número de conexiones que se deben atravesar desde un nodo para llegar a otro. En caso de que los nodos no pertenezcan al mismo subgrafo se considera que su distancia es infinita. También definimos que la distancia de un nodo a si mismo es siempre cero. De esta manera podemos definir la matriz de distancias, con 0 en la diagonal principal.

Distribución de Poisson

Cuando trabajamos en una red aleatoria (de una probabilidad P) con un número suficientemente grande de nodos (N), podemos observar que la probabilidad de que un nodo tenga un numero K de conexiones se puede aproximar por una función de Poisson tal que:

$$P(K) = \frac{e^{-a}a^k}{k!}$$

Siendo $a = P(N-1)$.

Se conoce el valor a como conectividad media de la red.

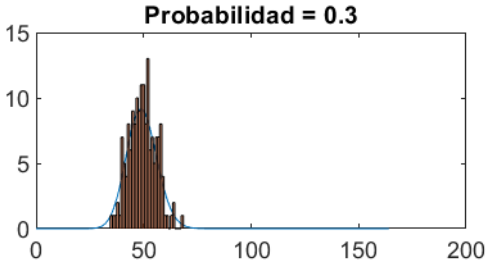

Ilustración 1 Distribución de Poisson comparada con el número de conexiones

Probabilidad crítica

Cuando estudiamos la evolución de las redes aleatorias en función de la probabilidad de unión se puede observar que hay una probabilidad crítica. De manera que una vez la probabilidad sea mayor que la probabilidad crítica, solo habrá un clúster que cubra todo el grafo. Se puede observar una transición de fase de clústeres aislados entre si a uno que contiene todos los puntos.

La probabilidad crítica solo se puede estimar con cierta seguridad tomando el límite cuando el número de nodos tiende a infinito. Al no poder trabajar con un número de nodos tan elevados de manera útil, hemos desarrollado programas que realizan una buena aproximación de la probabilidad crítica. Comentaremos estos programas más adelantes, pero cabe destacar que al ser experimentos aleatorios se puede llegar a dar que el programa falle en alguna ocasión. Esto sucede en general para todo el proyecto, pero afecta especialmente al estudio de la probabilidad crítica

Coeficiente de clustering

Otra propiedad muy importante en el estudio de grafos es el coeficiente de clustering. Nos indica cómo están conectados, de media, los vecinos de un nodo entre ellos. Para su estudio es importante definir el coeficiente de un solo nodo con el que se realiza el promedio. Para su cálculo utilizaremos la fórmula:

$$C_i = \frac{2n_i}{k_i(k_i-1)}$$

Donde n es el número de conexiones entre los k vecinos de un nodo. El máximo número de conexiones que puede haber entre los k vecinos de un nodo sería:

$$\frac{k_i(k_i-1)}{2}$$

Para calcular el coeficiente promedio, lo único que necesitamos hacer es la media aritmética de todos los coeficientes individuales.

$$[\![C]\!] = \frac{1}{N}\sum_{1}^{N} C_i$$

Una propiedad interesante, es que, para grafos suficientemente grandes, podemos comprobar que este coeficiente también se puede calcular de la forma:

$$[\![C]\!] = \frac{a}{N}$$

Donde a es la conectividad media de la red usada anteriormente.

Resultados experimentales

Programas básicos

Uno de los comandos imprescindibles para la realización de este proyecto es el comando “graph”, así que intentaremos explicar en profundidad todas las opciones que nos aporta.

A través del comando “graph” se genera un grafo unidireccional a partir de una matriz simétrica, la matriz de adyacencia, o un conjunto de dos vectores. Lo que quiere decir que el grafo sea unidireccional es que, si un nodo A está conectado a un nodo B, este último está conectado también al nodo A, mientras que en los grafos

bidireccionales una cosa no implica la otra. En caso de querer centrarnos en la creación de grafos bidireccionales, utilizaremos el comando "digraph" que funciona de manera igual a este pero la matriz que lo crea no tiene por qué ser simétrica.

A los nodos de un grafo se les puede dar propiedades, ya sean nombres, valores numéricos, etc. Se puede utilizar esta propiedad para hacer referencia a uno de estos nodos en caso de que queramos cambiar las propiedades de este o para llevar el grafo a la realidad.

Se puede utilizar el comando "plot" para representar de una manera visual el grafo. Esto sobre todo tiene importancia en los primeros momentos de la programación, para poder comprobar que los resultados que se obtienen concuerdan con el grafo que hemos creado. Además, como nuestro proyecto se basa en estudiar propiedades con gran carácter visual, el representar el grafo nos ayuda en gran parte a entender estas propiedades.

Es importante resaltar que sin el comando "graph" este trabajo resultaría mucho más difícil y desde luego mucho más complicado de entender ya que muchos de los comandos de los que dispone Matlab se derivan a partir de un grafo generado con "graph". Con esto empezamos a intuir el gran desarrollo que contiene Matlab para el estudio de los grafos.

Programa PruebaAuto

En primer lugar, procedemos a estudiar la importancia de los autovalores con respecto a los grafos. Para ello, creamos mediante bucles varios grafos con distintas probabilidades, utilizando la matriz de adyacencia. Una vez fabricado el grafo, calculamos la matriz de grado y la laplaciana, ya descritas anteriormente. Posteriormente, sacamos los autovalores de la laplaciana y contamos cuántas veces encontramos el autovalor 0.

Como ya hemos comentado antes, la multiplicidad de este autovalor nos permite saber cuántos subgrafos o clústeres hay. Para verlo de una manera más clara, representamos los subgrafos, mostrando en pantalla la probabilidad con la que se ha formado el grafo y el número de autovalores 0.

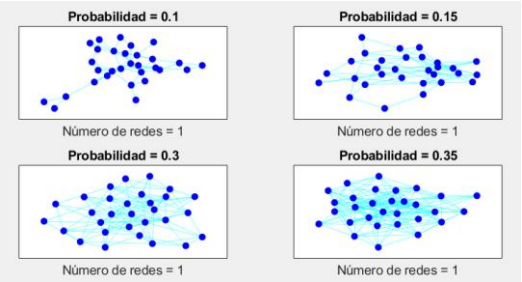

Ilustración 2 Plot grafos estudiados

En este programa comenzamos a observar el fenómeno de la probabilidad crítica, por lo que, usando una función que veremos más adelante, la calculamos y la representamos en pantalla.

```
Probabilidad límite aproximada = 0.206
```

Ilustración 3 Ejemplo de Probabilidad límite calculada

Función Auto

Como apreciamos en la figura, la multiplicidad del autovalor 0 nos será útil para estimar una probabilidad crítica, a partir de la cual el autovalor 0 sólo aparecerá una vez. Por eso, decidimos crear una función para, sabiendo los nodos y el número de veces que se quiere realizar el grafo (para variar la probabilidad), averiguar los autovalores 0 del grafo. En los argumentos de salida también incluimos la probabilidad.

Programa PruebaPoisson

En este programa se busca mostrar de manera gráfica cómo el número de uniones por nodo se aproxima por la función de Poisson. Para ello, generamos grafos con ciertas probabilidades, y se representa la función de Poisson en función de esta probabilidad, como ya se ha explicado anteriormente. Después se ajustan las

gráficas para que tengan una altura similar y se puedan comparar de manera más efectiva y sencilla. Las representamos todas dentro de un subplot.

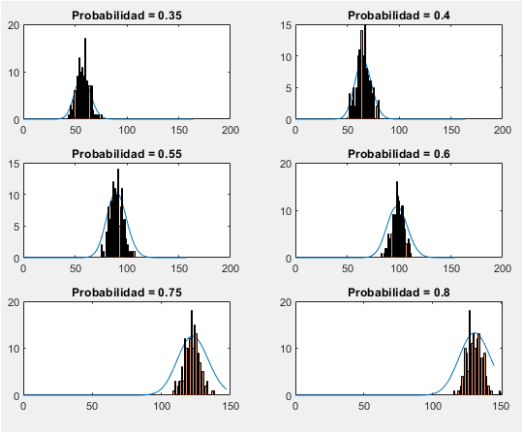

Ilustración 4 Distribución de Poisson y conectividad del grafo

Programa Probabilidad_Critica

A continuación, comenzamos a estudiar una de las características más interesantes, pero más difíciles de estudiar de los grafos aleatorios. Al no haber una fórmula o método lo suficientemente correcto para estudiarlo, decidimos hacer una aproximación de la probabilidad crítica mediante la repetición de grafos, viendo a partir de que probabilidad el autovalor 0 comienza a aparecer solo una vez. La mejor forma de realizarlo es mediante una función, que comenzamos a describir.

Función pc

En esta función nos valemos de la función autos descrita anteriormente. Los argumentos de entrada son el número de nodos, la veces que se quiere repetir el grafo (para variar la probabilidad) y las veces que repetimos el proceso. Dentro de la función lo que hacemos es averiguar cuál es la primera probabilidad a partir de la cual el autovalor 0 aparece solo una vez. Repetimos ese proceso y calculamos la media, parámetro que devolvemos junto al vector de probabilidades críticas.

Posteriormente, aplicamos en un programa esta función un número de veces. Nos quedamos con el máximo de todas las probabilidades críticas que hemos sacado de la función. Representamos un grafo creado con una probabilidad un poco mayor a la calculada y vemos el número de redes de las que dispone.

```
Probabilidad límite calculada = 0.326
```

Ilustración 5 Ejemplo de Probabilidad límite calculada

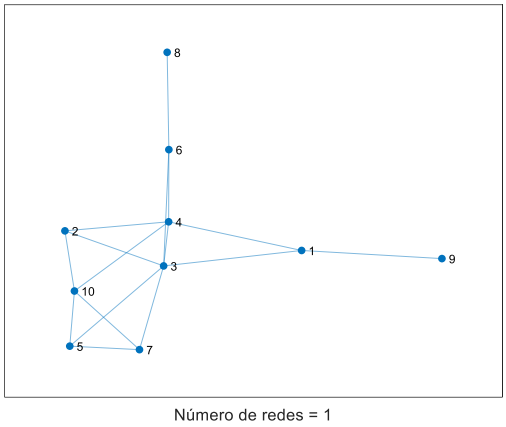

Ilustración 6 Red calculada con probabilidad = pc+0.05

Cabe destacar que al ser probabilidades puede darse el caso de que, al generar el grafo con una probabilidad, éste tenga solo una red y que el siguiente generado con una probabilidad un poco mayor también tenga una red, pero que ninguna de estas dos sea la probabilidad crítica. Como en el método que utilizamos esto puede llegar a suceder, hay veces que la probabilidad que obtenemos no es la crítica, por lo que, al generar el grafo, éste llega a tener dos redes. En nuestros experimentos, esto ha ocurrido alrededor de un 5-6% de las veces. Para minimizar este error se tienen 2 opciones:

-Aumentar el número de repeticiones donde se lleva a cabo el proceso, lo que acarrea que el programa aumente mucho su duración ya que estaría basado en bucles y como sabemos Matlab es un lenguaje que tiene muchos problemas a la hora de tratar con bucles tan grandes.

-Aumentar el número de nodos, ya que la única fórmula matemática que aproxima la probabilidad crítica se da cuando el número de nodos tiende a infinito.

Programa Diametro

Pasamos a describir otra de las características de los grafos, su diámetro. Para ello creamos varios grafos con la misma probabilidad, pero distinto número de nodos. Con los grafos creados, calculamos el diámetro mediante la siguiente fórmula:

$$D = \frac{\ln N}{\ln K}$$

Siendo $K = PN$. Almacenamos los valores de los diámetros y de K. Una característica destacada de K es:

- Si es menor que 1, el grafo está compuesto por árboles aislados y su diámetro es igual al de un árbol.
- Si K es mayor que 1 hay una agrupación gigante. El diámetro es igual que el diámetro de la agrupación gigante si K es mayor o igual a 3.5 y es proporcional a D.
- Si K es mucho mayor que $\ln N$ casi todo el grafo está conectado y el diámetro es más o menos D.

Para observar mejor esta propiedad representamos los grafos indicando el número de nodos y los valores de K.

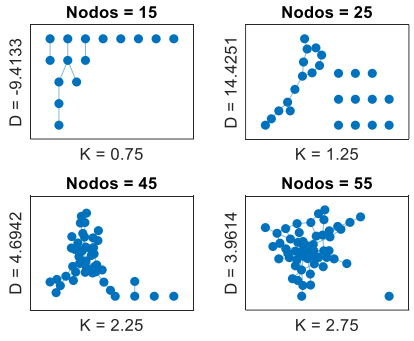

Ilustración 7 Representación K y diámetros

Programa grado de separación (GS)

Este programa nos servirá para estudiar una característica de los grafos, el grado de separación. El grado de separación se define como la media de la distancia que hay entre dos puntos del grafo. Para estudiarlo creamos un grafo con un número de nodos y probabilidad ya definidos. Una vez creado, nos valemos del comando "shortestpath" para averiguar la distancia más corta entre dos puntos del grafo.

Si la distancia es un número finito, lo vamos almacenando en un vector para luego realizar la media, que definiremos como grado de separación.

Si la distancia es infinita, ocurre cuando dos nodos no están conectados, almacenamos este "infinito" en un vector, para luego tener en cuenta el número de nodos cuya distancia entre sí es infinita. Por último, mostramos en pantalla el grado de separación y el número de "infinitos".

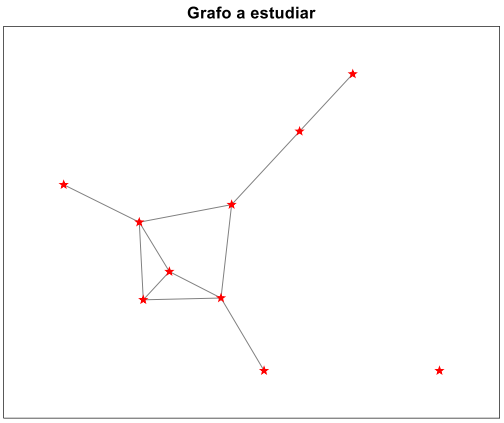

Ilustración 8 Representación grafo que se estudia

```
Grado de separación = 1.8889
Número de distancias infinitas = 9
```

Ilustración 9 Ejemplo de un grafo de 10 nodos con 1 inconexo

Programa Vecinos

Nos disponemos a estudiar el coeficiente de clustering o clusterización. Para ello, definimos una serie de parámetros: nodos, probabilidad, asignación de vectores… y

creamos un grafo. Como estudiamos un grafo único y para hacerlo más interesante, cogemos un gran número de nodos con una probabilidad de unión muy baja. Una vez creado el grafo, valiéndonos del comando "neighbors" averiguamos los vecinos de cada nodo y vemos si están conectados. Con esto, calculamos el coeficiente de clustering individual. De ahí averiguamos el coeficiente de clustering general y lo comparamos con el deducido de la fórmula antes explicada, viendo que coincide con gran exactitud.

```
Coeficiente de clustering general numérico = 0.099575
Coeficiente de clustering general fórmula = 0.0998
```

Ilustración 10 Ejemplo de un grafo de 500 nodos y probabilidad 0.01

Programa PruebaCiudades

En este programa pasamos a estudiar algunos de los muchos comandos de los que dispone Matlab para los grafos. Para hacer el estudio más familiar, nombramos los nodos como estaciones de metros que tienen una probabilidad definida de estar unidas. Creamos el grafo mediante la matriz de adyacencia y nos disponemos a usarlo para los diferentes comandos. La mayoría de ellos permiten ser representados en una figura por lo que haremos directamente o extraeremos un subgrafo para representarlos. En algunas figuras incluiremos algunas características especiales como marcar la distancia en el grafo, marcar una trayectoria dentro de un grafo mayor o cambiar el color y dibujo de los nodos

El primer comando será "neighbors", que nos permite ver cuáles son los "vecinos" de un grafo.

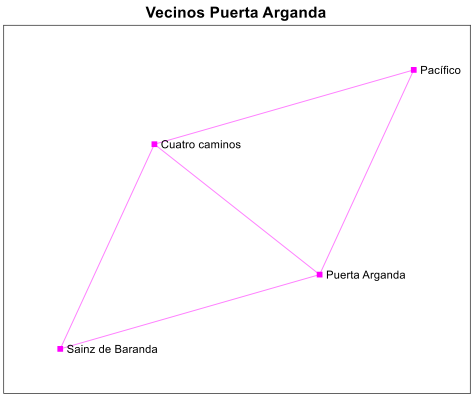

Ilustración 11 Comando Neighbors

El siguiente es el comando "shortestpath", ya utilizado anteriormente, que nos permite averiguar el camino más corto entre dos nodos, pudiendo averiguar el vector con los nodos o la distancia, entendiéndose como la hemos escrito anteriormente.

Ilustración 12 Comando Shortestpath

El comando "distances" nos permite averiguar la distancia entre dos puntos o bien la matriz de distancias, con las distancias entre todos los puntos.

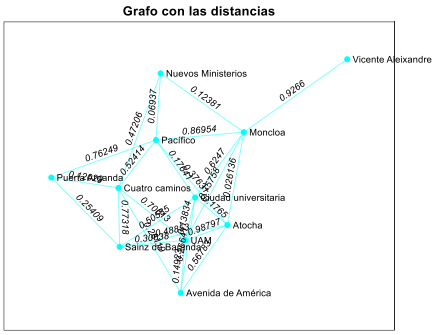

Ilustración 13 Comando Distances

Ahora pasamos a dos comandos similares: "minspantree" y "shortestpathtree". Ambos nos dibujan un árbol, pero el primero lo hace con las posibles trayectorias desde el primer nodo, mientras que el segundo se centra en los caminos más cortos desde un punto definido.

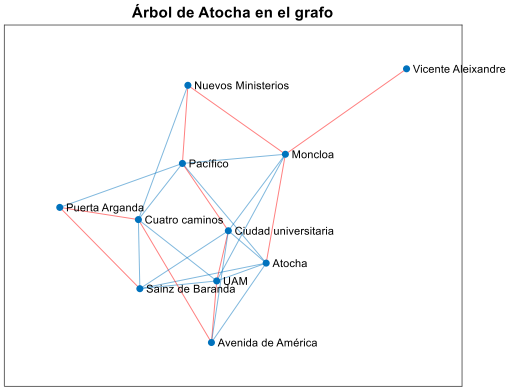

Ilustración 14 Comando Minspantree

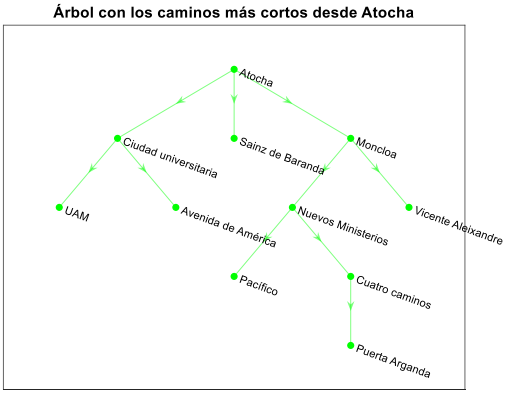

Ilustración 15 Comando shortestpathtree

El comando "degree" nos proporciona un vector con el número de conexiones de cada nodo, esta característica es conocida como "grado del grafo".

Por último, tenemos el comando "nearest", a partir del cual podemos saber que nodos están a una distancia inferior definida de un nodo, y el comando "edgecount" que cuenta el número de conexiones entre dos puntos dados.

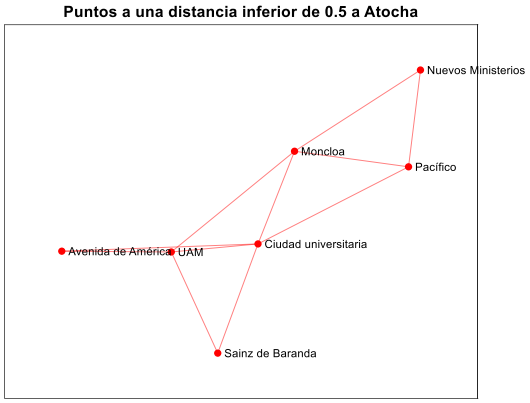

Ilustración 16 Comando nearest

```
Distancia entre UAM y Puerta Arganda = 0.1388
Número de conexiones entre Atocha y Pacífico = 1
```

Ilustración 17 Ejemplo de representación de datos de ciudades

Programa RedCiudad

Nos disponemos a estudiar un problema realista basado en la creación de una red de transportes en una ciudad con una probabilidad aleatoria de unión. Para ello, y para que sea más familiar, definimos una serie de nodos a los que les ponemos un nombre relacionado con la situación de la parada. También definimos el número medio de conexiones que queremos que tenga cada nodo. A continuación, nos disponemos a crear varios grafos con distintas probabilidades y calculamos la media de conexiones valiéndonos del comando "degree". Este proceso lo repetimos varias veces, seleccionando la probabilidad a partir de la cual el número medio de conexiones es mayor o igual al pedido. Para verlo de una forma más gráfica, representamos el número medio de conexiones en función de la probabilidad, marcando con una línea el número pedido.

Utilizamos el comando “subplot” para ver este gráfico cada vez que lo repetimos.

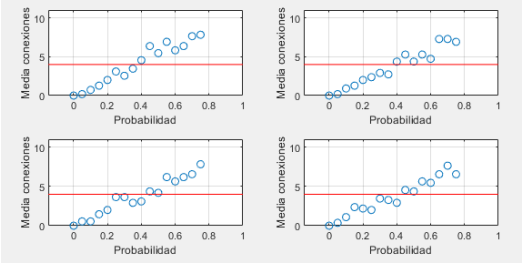

Ilustración 18 Número medio de conexiones en función de la probabilidad

Por último, hacemos la media de las probabilidades “críticas” que hemos ido obteniendo y también sacamos la probabilidad máxima, con la que con más seguridad cumpliremos el requisito. Usaremos esta probabilidad para crear un grafo y ver que efectivamente, cumple con lo pedido.

```
Probabilidad media a partir de la cual sucede = 0.40625
Probabilidad más segura para la que sucede = 0.55
Número medio de conexiones para p =0.55 ==> 5.4545
```

Ilustración 19 Representación de los valores

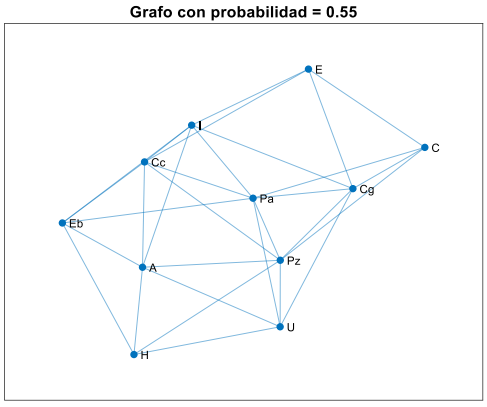

Ilustración 20 Grafo del problema

Función viajes y programa PruebaViajes

Viendo la gran utilidad de Matlab para el tratamiento de grafos, nos surgió la idea de crear una función que nos permitiese averiguar el camino más corto para realizar un viaje desde un punto a otro.

Para estudiarlo, creamos una función cuyos argumentos de entrada son una lista de ciudades, una lista de medios en los que viajar, tres vectores con distancias, costes y tiempos respectivamente, y el punto inicial y final.

El primer paso de la función es crear el grafo, para ello creamos mediante bucles dos vectores con los que definimos que los puntos estén unidos. Una vez creados los vectores, usamos el comando “graph” para crear el grafo, donde introducimos los vectores creados anteriormente, el vector con las distancias y la lista con las ciudades. A continuación, añadimos los medios, costes y tiempos. Valiéndonos del comando “simplify”, extraemos los diferentes subgrafos con los mínimos costes, tiempos y distancias. Estos tres subgrafos los incluiremos en los argumentos de salida de la función. Por último, extraemos las distancias mínimas en los subgrafos de tiempos y costes entre los puntos iniciales y finales definidos anteriormente. Representamos estas dos trayectorias en pantalla.

Aplicamos esta función en el programa llamado “PruebaViajes”, definiendo 10 ciudades y 5 medios, con costes, tiempos y distancias aleatorios.

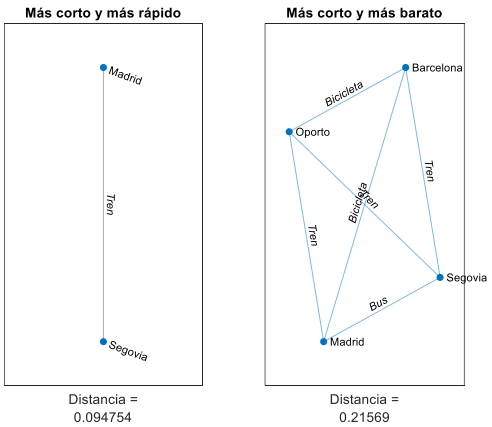

Ilustración 21 Plot del camino más corto y del camino más barato.

Función viajesd y programa PruebaViajesD

El mismo razonamiento lo aplicamos ahora, pero incluyendo la condición del que punto inicial y el final no pueden estar unidos

directamente, teniendo que atravesar una red de ciudades o puntos para llegar de un punto al otro. Esto implica un poco más de complicación a la hora de generar los vectores que nos permiten generar el grafo. Una vez solucionado este problema, seguimos el mismo procedimiento para simplificar el gráfico y representar las trayectorias más cortas y rápidas y más cortas y baratas.

Aplicamos esta función en el programa llamado "PruebaViajesD", definiendo 14 ciudades y 5 medios, con costes, tiempos y distancias aleatorios.

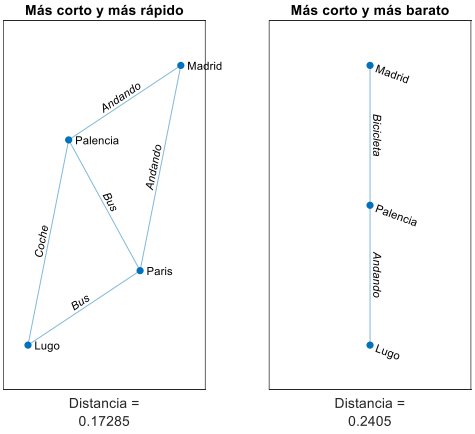

Ilustración 22 Plot del camino más corto y del camino más barato.

Discusión

En este proyecto hemos estudiado con gran éxito las propiedades más básicas de los grafos aleatorios. Hemos comprobado con bastante acierto las posibles fórmulas sobre algunas características del grafo. Además, hemos indagado en algunas de las opciones que presenta Matlab para el uso de los grafos y hemos trasladado lo aprendido para la resolución de algunos problemas realistas.

Conclusiones

Los grafos aleatorios nos proporcionan un gran abanico de posibilidades y aplicaciones. Además, Matlab se encuentra muy optimizado para el uso de grafos, con diversos comandos integrados en su código que serían difícil de programar, lo que lo hace una excelente herramienta para este trabajo. En este proyecto nos hemos centrado fundamentalmente en el estudio de algunas de las características más importantes de los grafos aleatorios, pero, aunque estas sean las más fundamentales, podemos encontrar una diversidad de propiedades muy interesantes.

Como ya hemos demostrado, los grafos se pueden estudiar de manera teórica, pero tienen un sinfín de aplicaciones realistas como hemos podido demostrar en el código de viajes.

Un gran ejemplo de esto sería, la aplicación de Google Maps, la cual estudia sus mapas y rutas a través de grafos de una manera parecida, aunque, por supuesto, mucho más sofisticada que la que hemos usado nosotros. Como sería prácticamente imposible crear a mano una red tan compleja como la de carreteras a lo largo de todo el mundo, el método que usan para crear la red es a través de información visual que captan por satélite.

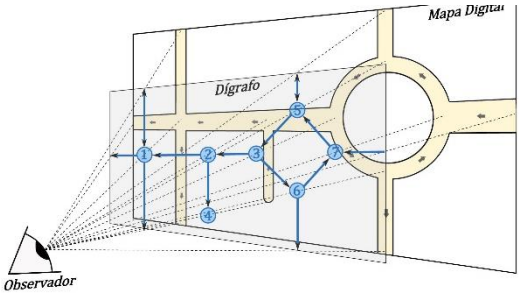

Ilustración 23 Ejemplo funcionamiento Matlab.

Como podemos ver el estudio de los grafos nos abre un nuevo mundo de estudio con muchas aplicaciones para el que Matlab resulta una gran herramienta.

Referencias bibliográficas

Dodds, P. (2011). *Applications of Random Networks*. Recuperado de: http://materias.df.uba.ar/IRCBSa20

16c2/files/2016/09/11_RedesAleatorias.pdf

Erdös, P., & Rényi, A. (1958). *On radom graphs I.* Recuperado de: (http://snap.stanford.edu/class/cs224w-readings/erdos59random.pdf

Ortega, J. L., López-Sauceda, J., Carrillo, J. G., & Sandoval, J. (2019). Método de construcción de dígrafos a partir de redes viales reales en mapas digitales con aplicaciones en la búsqueda de rutas óptimas. *Informes de la Construcción*, *71*(556), e320.

Othoniel, G. (2010). *Riesgo sistémico. Una aplicación de los grafos aleatorios.* (Tesina, Universidad de Guanajuato, México). Recuperado de: https://probayestadistica.cimat.mx/sites/default/files/PDFs/TE334CanoMoo.pdf

Sáenz, M. (2019). *Desempeño asintótico de algoritmos secuenciales en grafos aleatorios.* (Doctoral dissertation, Universidad de Buenos Aires). Recuperado de: http://bibing.us.es/proyectos/abreproy/11749/fichero/Volumen+I%252FP1-5-GRAFOS+REGULARES+Y+ALEATORIOS.pdf

Anexo

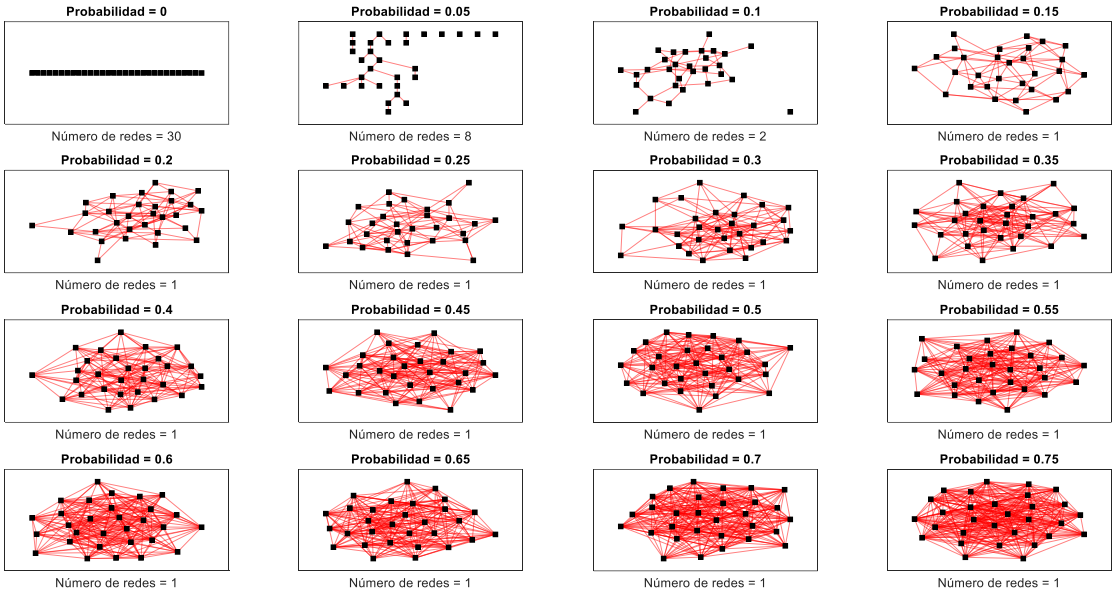

Ilustración 24 Plot completo programa prueba autos

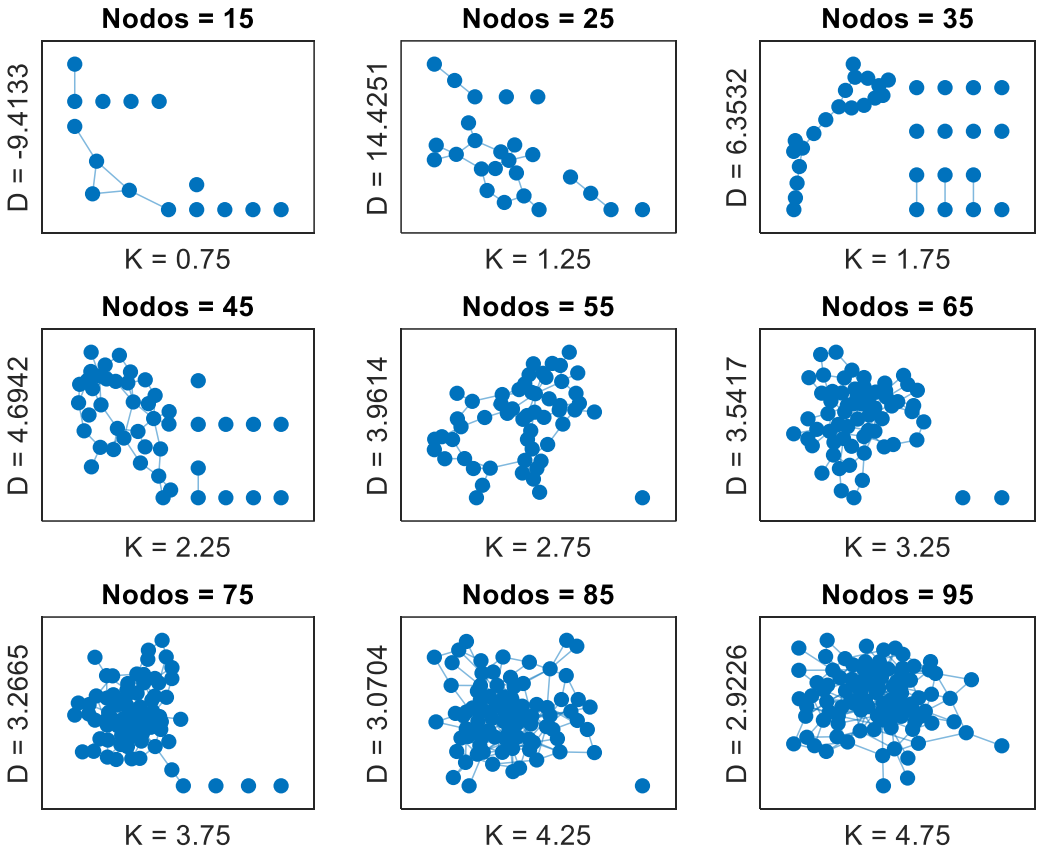

Ilustración 25 Plot completo programa diametro

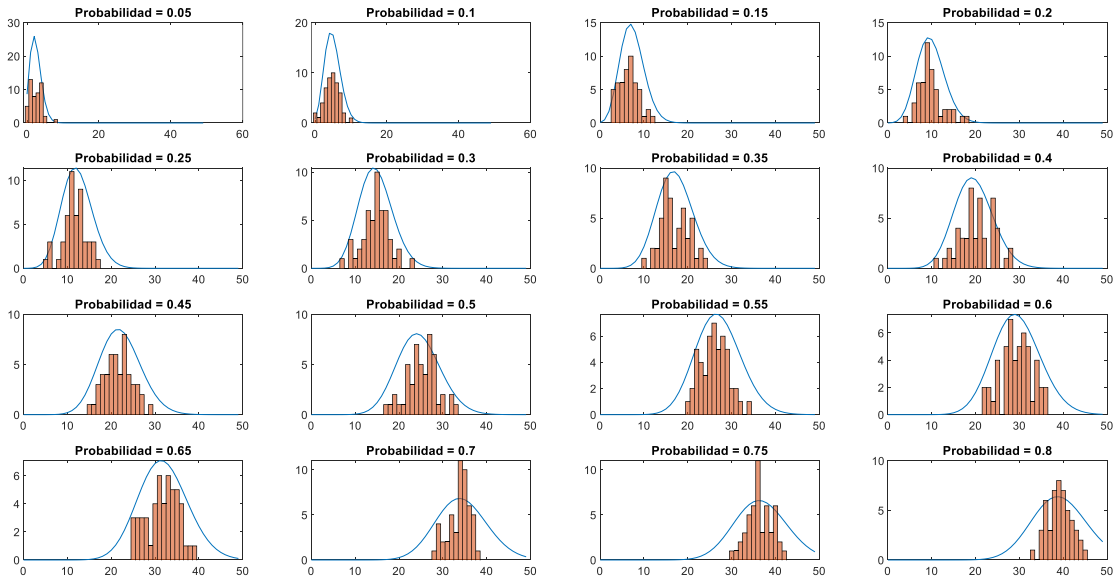

Ilustración 26 Plot completo programa Poisson

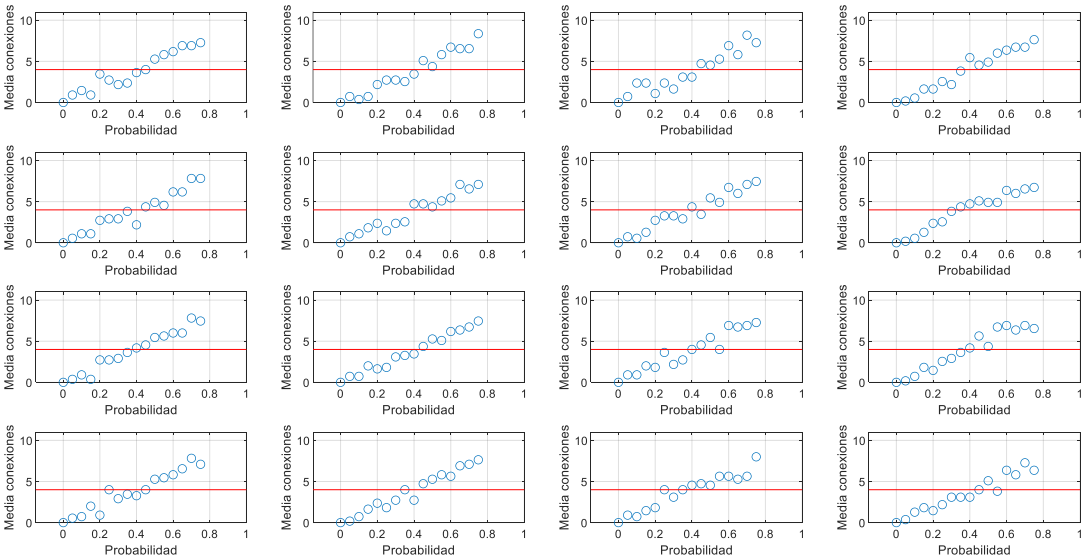

Ilustración 27 Plot complete RedCiudad.