

Transición de fase en propiedades de grafos

Memoria del Proyecto de A

David Minguez, Adrián Lozano y Alex Iniesta

Cuatrimestre de Primavera

13/05/2019

Índice

1. Introducción	3
2. Modelos	4
2.1. Cuadrícula	4
2.2. Modelo de Erdős–Rényi	5
2.3. Modelo de Watts-Strogatz	6
2.4. Grafo uniforme	7
2.5. Grafo cíclico	7
3. Propiedades	8
3.1. Top Bottom	8
3.2. Cíclico	8
3.3. Conexo	9
3.4. Grafo Euleriano	9
4. Experimentos	10
4.1. Top and Bottom en cuadrícula	11
4.1.1. Idea de resolución	11
4.1.2. Percolación por vértices	13
4.1.2.1. Coste	15
4.1.3. Percolación por aristas	16
4.1.3.1. Coste	17
4.1.4. Método MonteCarlo Vértices	17
4.1.4.1. Experimento	18
4.1.4.2. Coste	19
4.2. Top and Bottom en grafos aleatorios	19
4.2.1. Método testTopBottom	20
4.2.2. Experimentación con Erdős–Rényi	21
4.2.2.1. Percolación por vértices	21
4.2.2.2. Percolación por aristas	22
4.2.2.3. Conclusiones	22
4.2.2. Experimentación con el modelo uniforme	23
4.2.3. Experimentación con el modelo de Watts-Strogatz	24
4.2.3.1. Percolación por nodos	24
4.2.3.2. Percolación por aristas	25
4.2.3.3. Conclusiones	25
4.3. Grafos cíclicos	26
4.3.1. Método testCycle	26
4.3.2. Experimentación en modelo cíclico	26

4.3.2.1. Percolación por vértices	26
4.3.2.2. Percolación por aristas	27
4.3.2.4. Conclusiones	28
4.3.2. Experimentación en modelo de Watts-Strogatz	28
4.3.2.1. Percolación por vértices	28
4.3.2.2. Percolación por aristas	29
4.3.2.3. Conclusiones	29
4.4. Grafos conexos	29
4.4.1. Búsqueda en profundidad (DFS)	30
4.4.2. Experimentación usando el modelo de Erdos	30
4.4.3. Experimentación usando el modelo uniforme	31
4.5 Grafos Eulerianos	32
4.4.1 Método de comprobación	33
4.4.2 Ciclo Euleriano	33
4.4.3 Camino Euleriano	34
5. Conclusiones	35
6. Bibliografía	36

1. Introducción

El objetivo de esta práctica es conocer y aprender diferentes modelos de grafos aleatorios y estudiar las propiedades de grafos más explotables para cada modelo.

Se presenta el concepto de percolación, que consiste en, a partir de una probabilidad q , decidir si un vértice o arista falla o, lo que es lo mismo, que con probabilidad $1-q$ no falla.

La idea es aplicar este proceso en grafos que cumplan cierta propiedad y ver a partir de qué probabilidad de percolación, esta propiedad se cumple -o no se cumple-. A ese valor de probabilidad le llamamos transición de fase.

Hemos generado los modelos de grafos aleatorios de Erdős–Rényi i Watts-Strogatz y generado, partiendo del de Erdős–Rényi, el modelo uniforme, y además un modelo de grafo cíclico aleatorio, todos ellos comentados en el siguiente apartado.

Para esta práctica hemos estudiado las propiedades de *top and bottom*, conexo, grafo euleriano y grafo cíclico. Más adelante se explica en profundidad cada una de ellas y cómo las hemos implementado.

También hemos creado la clase Union-Find que implementa Weighted Quick Union con path compression. Dicha clase contiene su .hh y su .cc. La hemos creado porque la hemos considerado útil para probar propiedades como conectividad, encontrar ciclos y la propiedad de *top and bottom* que se presenta en el enunciado. La implementación es la habitual y que se ilustra en las diapositivas facilitadas por el enunciado.

2. Modelos

Todos los modelos se han agrupado en el fichero *graphgen.cc*, excepto la cuadrícula, que la generamos al mismo tiempo que testamos por tal de mejorar la eficiencia. La implementación para grafos aleatorios la hemos realizado con una matriz de adyacencias. Cada modelo tiene un método asociado que retorna un grafo aleatorio basándose en dicho modelo.

2.1. Cuadrícula

El modelo de cuadrícula consiste en un grafo cuadrado de tamaño N por N donde cada nodo está enlazado a su anterior, al siguiente y al que tiene arriba y abajo de la siguiente forma.

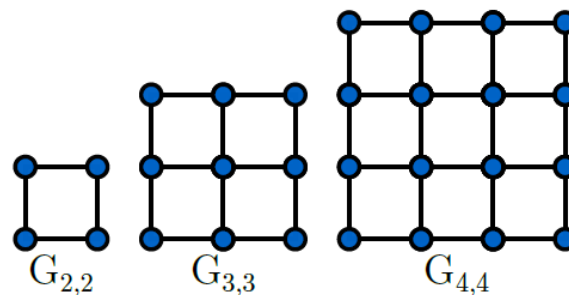


Figura 2.1. Ejemplo de cuadrículas de diferentes dimensiones

En esta primera aproximación, utilizaremos para representar la cuadrícula un grafo que estará implementado en un vector de tamaño $N \times N$. Hemos decidido no utilizar una matriz de adyacencias porque vemos muy ineficiente guardar la información en una matriz que tendría un tamaño de N^4 y conlleva malgastar mucho espacio ya que como mucho cada nodo puede estar conectado con cuatro vértices. Por otro lado, se podría optar por la representación de una lista de adyacencias que sería más eficiente, pero cómo sabemos con anterioridad cómo están definidas las conexiones en la graella (cada nodo con el de arriba, izquierda, derecha y abajo como mucho) vemos mucho más simple utilizar un vector de *structs* que contendrá la información para representar una estructura de *union-find* que aplicaremos para observar la propiedad que se pide en el apartado b.

La *struct* contiene la siguiente información:

- *parent*: El nodo representante
- *rank*: El ránking del nodo, aplicaremos *link by rank*
- *posGraella*: La posición exacta que ocupa en la cuadrícula: por ejemplo (0,0),(2,0),(3,3)
- *abierto*: Indicando si ese nodo está abierto o no (en el caso de percolación por aristas trataremos todos los nodos como abiertos)

2.2. Modelo de Erdős–Rényi

El modelo Erdős–Rényi es uno de los empleados en la generación de grafos aleatorios y, actualmente, se usa como base en la generación de otras redes. Existen, principalmente, dos variantes en la generación del modelo:

La primera recibe el número de vértices N y de aristas M se escoge de forma uniforme uno de entre todos los posibles grafos que existen con N vértices y M aristas.

El segundo, que es el que hemos implementado en el proyecto, recibe el número de vértices N y una probabilidad p . La probabilidad p define si incluir o no una arista en el grafo o, expresado de otro modo, todos los grafos con N vértices y M aristas tienen la misma probabilidad, $p^M(1-p)^{\binom{n}{2}-M}$. Otra forma de entenderlo sería que, a mayor probabilidad p , el modelo incluirá con mayor frecuencia grafos con más aristas. El número esperado de aristas es de $\binom{n}{2}p$. En la figura 2.2 se puede observar un ejemplo de posibles grafos obtenidos mediante el modelo.

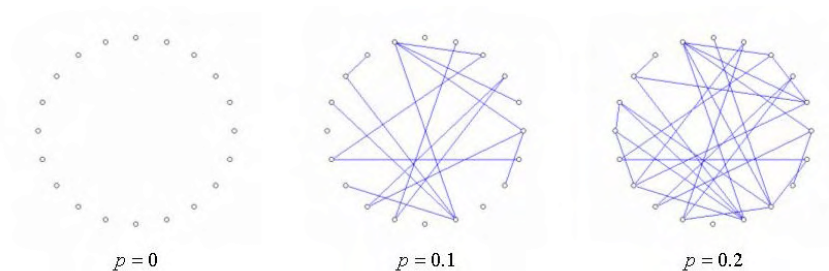


Figura 2.2. Evolución del modelo de Erdős–Rényi en función de p

Dado que nuestra implementación general de grafos se hace mediante una matriz de adyacencias, el coste de generar un grafo siguiendo el modelo de Erdős–Rényi es de $O(n^2)$, ya que tenemos que mirar cada posible arista entre dos vértices (en total, $n(n-1)/2$) y decidir si añadirla o no con probabilidad p .

2.3. Modelo de Watts-Strogatz

El modelo de Watts-Strogatz es un modelo de generación de grafos aleatoria que da como resultado grafos con propiedades de las llamadas *redes de mundo pequeño*. La principal propiedad que definen las *redes de mundo pequeño* es la de que la mayor parte de nodos no son vecinos entre sí, pero la mayoría de nodos pueden ser alcanzados desde otro nodo origen en un número pequeño de pasos. Por esta característica, podemos intuir que estamos hablando de grafos usualmente conexos, cíclicos y con un alto coeficiente de agrupamiento (cuán interconectado está un nodo con sus vecinos). En la figura 2.3 adjuntamos un ejemplo de grafo generado mediante el modelo de Watts-Strogatz.

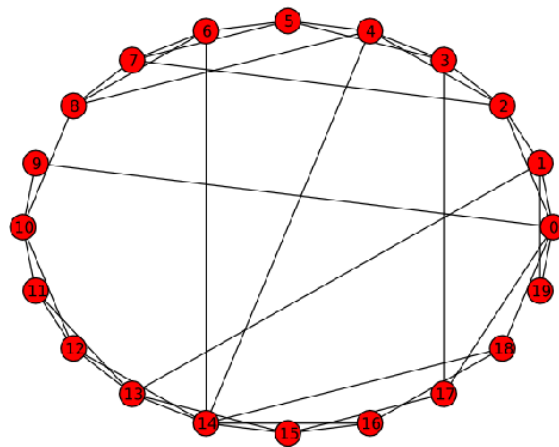


Figura 2.3. Ejemplo de grafo generado mediante el modelo de Watts-Strogatz

La implementación para generar grafos que siguen este modelo recibe tres parámetros, n , que corresponde al número de vértices, k , que nos dice a cuantos vecinos es adyacente cada vértice y p , la probabilidad con la que una arista es reemplazada por otra aleatoria. El procedimiento consta de dos partes:

En la primera, hacemos que cada nodo sea adyacente a sus k vértices vecinos. Lo que tendrá un coste $O(n \cdot k)$. En el peor caso, $k = n-1$, por tanto el coste será $O(n^2)$.

En la segunda, para cada arista, con probabilidad p elegimos si cambiarla por otra o no. El coste de esta parte, al recorrer todas las aristas, es $O(n \cdot k)$ lo que, como en la primera parte, en el caso peor puede ser $O(n^2)$.

2.4. Grafo uniforme

El modelo uniforme lo hemos creado basándonos en la idea del modelo de Erdős–Rényi que recibe el número de aristas. Nuestro modelo consiste en decidir de forma aleatoria el número de aristas que tendrá nuestro grafo y, seguidamente, vamos seleccionando aristas de forma aleatoria y las añadimos en el caso de que no estén añadidas ya. Los resultados que esperamos de este modelo son los de generar grafos variados con una distribución uniforme. En este modelo hemos decidido que el número de vértices también se defina de manera aleatoria, por tanto, el coste de generar un grafo de este tipo será $O(M)$, donde M podrá valer desde 1 hasta $n(n-1)/2$, es decir, el número máximo de aristas de un grafo con n vértices, por tanto, el coste se puede expresar también de la forma $O(n^2)$, pero, dado que n es variable, no se puede garantizar un tiempo en concreto.

2.5. Grafo cíclico

El modelo cíclico genera un grafo con un número aleatorio de ciclos con, al menos, un ciclo. La forma de generarlo consiste en decidir aleatoriamente el número de ciclos que como mínimo tendrá el grafo resultante e ir generando ciclos partiendo de vértices aleatorios. El tamaño de esos ciclos también se decide de manera aleatoria. Es importante destacar que el número de ciclos decidido no encajará necesariamente con el número de ciclos que realmente tenga nuestro grafo resultante.

Este modelo lo hemos generado con la idea de aplicarle percolación y estudiar la transición de fase sobre la propiedad de si tiene ciclos.

El coste de generar este tipo de grafos es $O(n^2)$, donde n representa el número de vértices, pues como máximo generamos $n/2$ ciclos de tamaño como máximo n cada uno.

3. Propiedades

3.1. Top Bottom

Esta primera propiedad del enunciado del proyecto consiste en comprobar si hay conectividad entre partes específicas de un grafo.

En el caso de la cuadrícula, exactamente es la conectividad entre los vértices de la fila de arriba (*top*) y los vértices de la fila de abajo (*bottom*). En el caso de grafos aleatorios, hemos seleccionado ciertos vértices que podían resultar interesantes como top y bottom. Esta es una propiedad muy interesante, ya que se puede utilizar para modelar la conductividad de un material eléctrico o cuanto es de poroso un cierto material a la hora de que pase un líquido por él o no.

Para implementarlo, hemos decidido utilizar una estructura de *union-find* donde los conjuntos serán los diferentes elementos conexos de la cuadrícula y podremos saber si están conectadas la parte de arriba y abajo de la cuadrícula con una simple consulta de sus representantes (*find*).

En el caso de grafos generales, la implementación se ha tenido que enfocar de forma distinta, pues las adyacencias entre vértices son indeterminadas y no servía con explorar ciertas aristas. Para comprobar que se cumple la propiedad necesitamos acceder a todas las aristas y, por cada una realizar una operación de *union*. Finalmente, en la estructura del union-find tendremos dos elementos adicionales que se unirán con top y bottom respectivamente. Para ver si se cumple la propiedad bastará con ver si los representantes de estos dos vértices adicionales son el mismo.

3.2. Cíclico

Un grafo es cíclico si contiene algún ciclo. Hemos querido estudiar esta propiedad para ver cómo afecta la percolación sobre grafos que sabemos que tienen un ciclo y estudiar así la transición de fase, lo que significa, en este caso, a partir de qué valor de probabilidad de percolación pasa de ser cíclico a ser acíclico.

El algoritmo que hemos implementado para buscar ciclos en un grafo hace uso de la estructura de datos union-find y consiste en ir generando la estructura naturalmente, recorriendo la matriz de adyacencias del grafo y realizando la operación $\text{union}(i,j)$ por cada arista entre los vértices i y j hasta que dicha operación se vaya a realizar entre dos vértices

los cuales la operación *find* resulte en el mismo valor, es decir, pertenezcan ya al mismo componente conexo.

La comprobación de esta propiedad consta de una primera parte que es inicializar la estructura *union-find* ($O(n)$) y recorrer la matriz de adyacencias y realizar las operaciones de *find* y *union*, que en nuestro caso, gracias a la implementación *weighted QU with path compression*, cada operación tiene coste $O(\log(N))$, lo que resulta en un total de $O(M\log(n))$, puesto que esta operación se realiza por cada arista del grafo, y siendo n el número de vértices y M el número de aristas del grafo y este coste se da, por ejemplo, en los casos en que no haya ciclos. Destacar que, para comprobar esta propiedad, como los grafos sobre los que trabajamos son dirigidos, la matriz es simétrica, por tanto es importante mirar tan sólo el triángulo superior (sin la diagonal), por tal de no mirar la adyacencia reflexiva ni recíproca, lo que nos generaría falsos positivos.

3.3. Conexo

Por definición un grafo $G(V, A)$ es conexo si para todo $u, v \in V$ existe un camino de u hasta v ; siendo un camino una sucesión de vértices tal que entre vértices sucesivos existe una arista entre ellos.

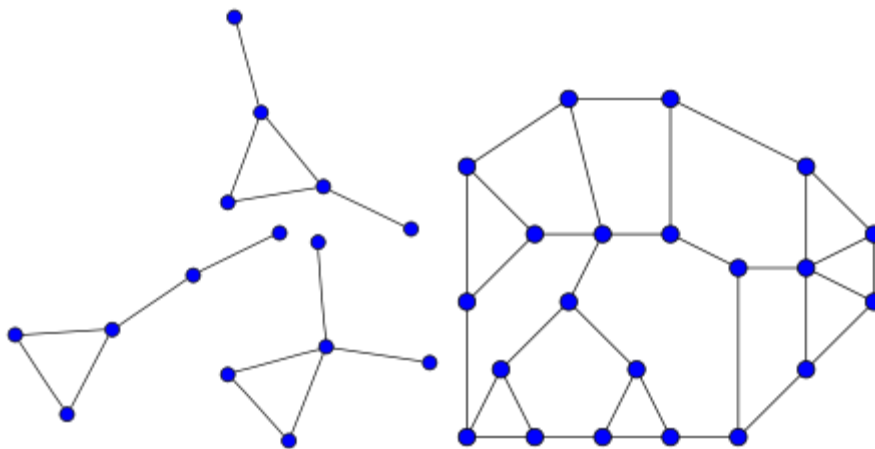


Figura 3.1. A la izquierda un grafo no conexo, y a la izquierda uno conexo

Existen una gran cantidad de algoritmos que pueden comprobar si un grafo es conexo, el utilizado por nuestra parte es un algoritmo de búsqueda en profundidad.

3.4. Grafo Euleriano

Un grafo Euleriano es todo aquel que contiene un ciclo Euleriano en él. Un ciclo Euleriano es un recorrido de vértices cerrado, no trivial, que no repite aristas y que además atraviesa todas las aristas del grafo.









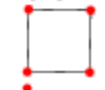





<i>singleton graph</i> 						
<i>2-empty graph</i> 						
<i>3-empty graph</i> 	<i>triangle graph</i> 					
<i>4-graph 6</i> 	<i>4-empty graph</i> 	<i>square graph</i> 				
<i>5-graph 8</i> 	<i>5-graph 14</i> 	<i>butterfly graph</i> 	<i>5-cycle graph</i> 	<i>5-empty graph</i> 	<i>(3,2)-fan graph</i> 	<i>pentatope graph</i> 

Figura 3.2. Grafos Eulerianos con $n = [1, 5]$ vértices

Existe un teorema que caracteriza los grafos Eulerianos, un grafo G es Euleriano si y sólo si:

1. G es conexo o todas sus componentes conexas menos una son vértices aislados (con grado igual a 0)
2. Todas las vértices de la componente conexa que no es isomorfa a un vértice aislado tiene grado par

Paralelamente a ciclo Euleriano podemos definir camino Euleriano, que es aquel recorrido que recorre todas las aristas sin repetirlas que no es cerrado.

4. Experimentos

En los experimentos que hemos realizado, por tal de ir acorde a las transparencias de referencia, hemos realizado los gráficos de forma que a más probabilidad de percolación, más aristas se incluyen en el grafo.

4.1. Top and Bottom en cuadrícula

En este apartado, correspondiente al apartado b del enunciado, estudiaremos la transición de fase en cuadrículas $N \times N$ bajo un proceso de percolación de nodos y uno de aristas. En concreto estudiaremos la propiedad de conectividad de la parte de arriba de la cuadrícula con la parte de abajo (*top bottom*).

El objetivo del experimento es encontrar un valor de transición de fase por el cual si aplicamos un proceso de percolación de nodos y aristas con probabilidad igual a ese valor, se cumple la propiedad de conectividad en la cuadrícula de modo que a partir de ese valor siempre se cumplirá la propiedad y con menos de ese valor nunca se cumplirá.

4.1.1. Idea de resolución

Para conseguir el valor que buscamos hacemos un bucle que ejecuta el programa con valores de probabilidad de percolación que van aumentando de 0,1 en 0,1 desde 0. Cuando detecta que se cumple la propiedad ejecuta otro bucle desde por ejemplo 0,5 a 0,6 aumentando de 0,01 en 0,01 para obtener un valor más preciso y conseguir un buen resultado.

Esto fue una primera aproximación, pero nos percatamos de que no se consigue un valor tan exacto. Esto es porque no existe un valor numérico preciso, si no que este valor de transición de fase indica una zona donde las probabilidades de que se cumplan la propiedad aumentan. Para ello haremos un cálculo de probabilidades ejecutando 100 veces con valores de 0,01 a 0,99 y viendo las veces que se cumple la propiedad o no, para tratar de visualizar la zona de transición.

Otra idea que hemos desarrollado, en el caso de percolación por nodos, es aplicar una **simulación Monte Carlo**.

El método Monte Carlo son una clase de algoritmos computacionales que se basan en muestras aleatorias para obtener valores numéricos. Principalmente se utiliza para aproximar expresiones matemáticas complejas y costosas de evaluar con exactitud. El

concepto que aplican es utilizar aleatoriedad para resolver problemas que son deterministas.

Aunque a primera vista no parece un método muy potente, pero proporciona soluciones aproximadas a una gran variedad de problemas matemáticos posibilitando la realización de experimentos a base de escoger números pseudoaleatorios en un computador.

Por eso ha sido utilizado mucho física y matemáticas, como por ejemplo, en 1948, Enrico Fermi, Nicholas Metropolis y Ulam consiguieron obtener estimadores para los valores característicos de la ecuación de Schrödinger para la captura de neutrones a nivel nuclear. También en la actualidad, por ejemplo, es parte fundamental de los algoritmos de *raytracing* para la generación de gráficos 3D.

Este método debe su nombre en referencia al Casino de Montecarlo en Mónaco porque es considerada la capital del juego, ya que sus características ruletas son un generador muy sencillo de números aleatorios.

Para este experimento, decidimos juntar los pasos de generación de la graella utilizando un proceso de percolación con la comprobación si está unida la parte de arriba con la de abajo. No lo separamos porque para este caso específico vemos mucho más práctico de implementarlo todo junto.

La idea que hemos aplicado es la siguiente. Al principio no hay ningún nodo creado y por lo tanto inicializamos todos como no abiertos. Para ver si está conectada la parte de arriba con la de abajo creamos un nodo ficticio *top* que estará conectado a todas las celdas de la parte de arriba que estén abiertas y lo mismo haremos con un nodo ficticio *bottom*.

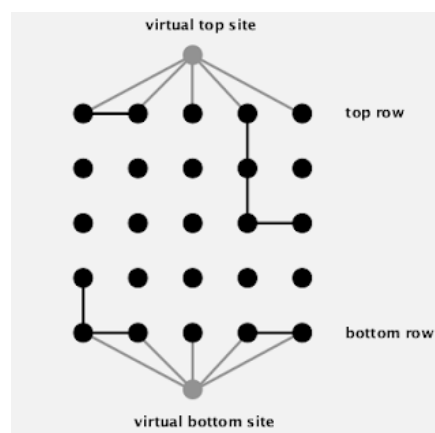


Figura 4.1.1 Representación de los nodos virtuales sobre la cuadrícula

A continuación recorreremos todos los nodos, y utilizando la probabilidad establecida decidimos si ponerlo en el grafo o no. En el caso de añadirlo, ahora falta explicar la forma de modelar las nuevas conexiones.

Como la conectividad entre sí se puede expresar en términos de conjunto, decidimos aplicar una estructura de *union-find* implementada con *path compression* y *link by rank*. Primero de todo, los nodos virtuales están abiertos y por cada nodo que decidimos abrir según nuestra función de probabilidad, lo marcamos como abierto y aplicamos un *union* para unirlo a los adyacentes que están abiertos (como mucho serán 4) y vamos aplicando este proceso en todo el recorrido.

Cada vez que se abre un nodo, observamos si el representante del nodo virtual de arriba es el mismo que el representante de abajo, en ese caso la propiedad se cumple y paramos la ejecución. En caso contrario, cuando el bucle que recorre toda la cuadrícula ha acabado y siguen sin estar conectados no se cumple la propiedad.

Esta manera es mucho más eficiente que hacer una búsqueda en profundidad en cada iteración para comprobar si desde alguno nodo de la parte inferior se puede llegar a la parte superior.

4.1.2. Percolación por vértices

Para empezar, realizamos un experimento con una N pequeña y la vamos aumentando de 100 en 100 para ver la tendencia del valor de transición de fase que encuentra el algoritmo.

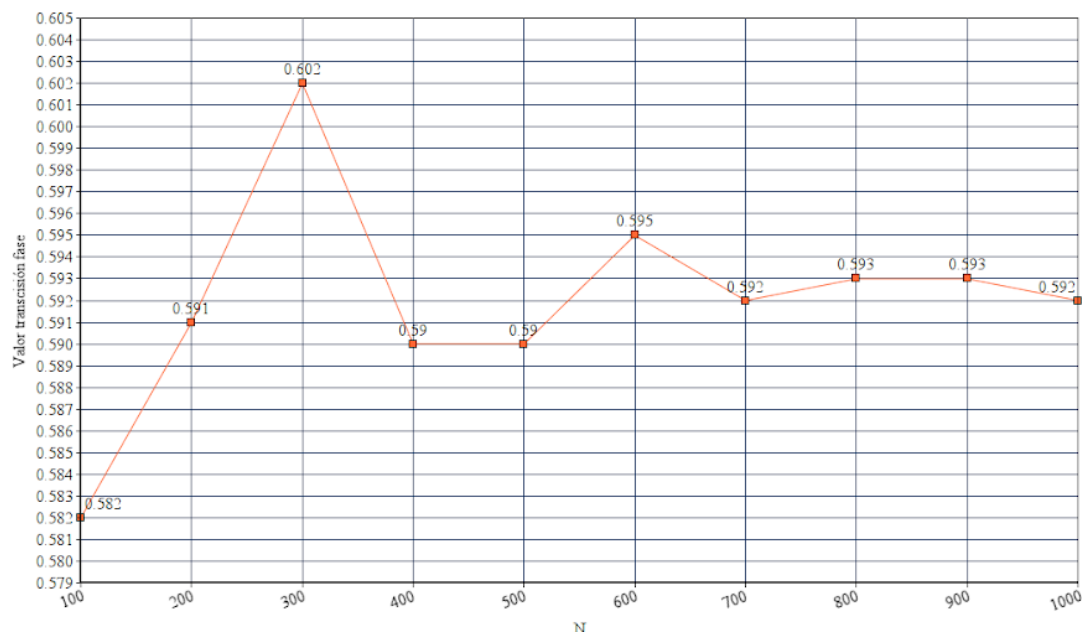


Figura 4.1.2.1 Gráfica de los valores de transición de fase encontrados por la primera aproximación

Como se puede observar en la gráfica, una vez la N es más grande de 700 el valor ya empieza a alcanzar 0,593 en todos los casos que es el resultado esperado.

A continuación, analizamos con un caso práctico el coste temporal del algoritmo. De forma parecida al experimento anterior, realizaremos una serie de ejecuciones empezando con una N=100 y aumentando de 100 en 100 hasta 1500. Ahora de en vez de tomar el tiempo en segundos será en milisegundos.

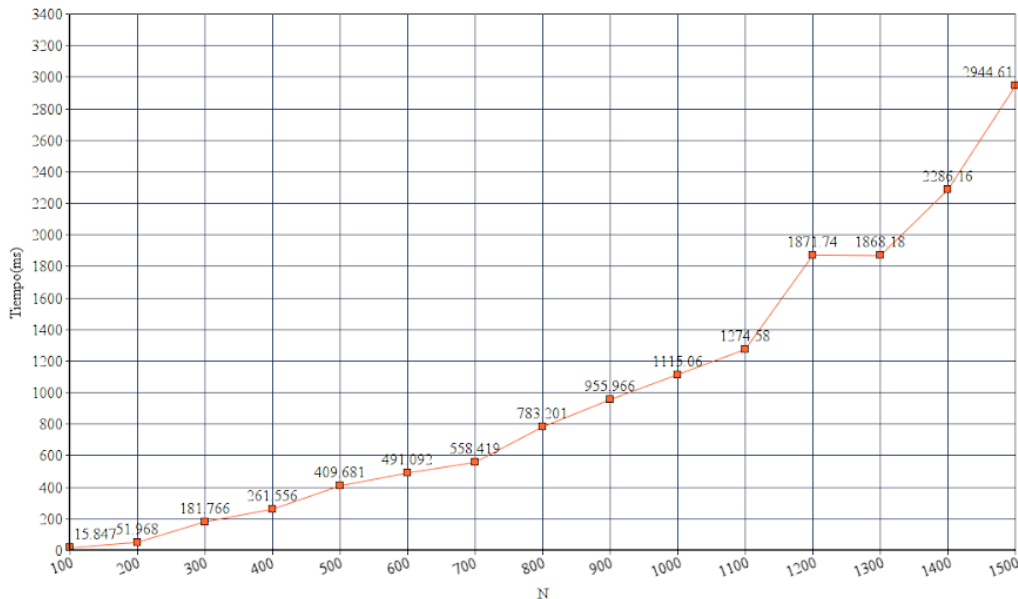


Figura 4.1.2.2 Gráfica del tiempo de ejecución en ms en función de N

Analizando el resultado, observamos que para valores pequeños de N el resultado parece lineal (ya que está muy próximo al lugar donde corta con el eje x) pero globalmente tiene aproximadamente crecimiento cuadrático como explicamos en su análisis correspondiente más adelante.

Esto fue el resultado que da la primera aproximación y vemos que los valores se acercan al resultado esperado pero no son exactos, esto viene dado por la zona que explicamos antes ya que estos valores se encuentran en la zona donde tienen más probabilidad de cumplir la propiedad. A partir de una N más grande, observamos que siempre da el mismo valor ya que la zona se estrecha y entonces hay menos valores con una probabilidad intermedia entre 0 y 1, y se obtiene el valor que buscamos.

Por eso, ahora realizamos el experimento para encontrar esta zona con una N=100.

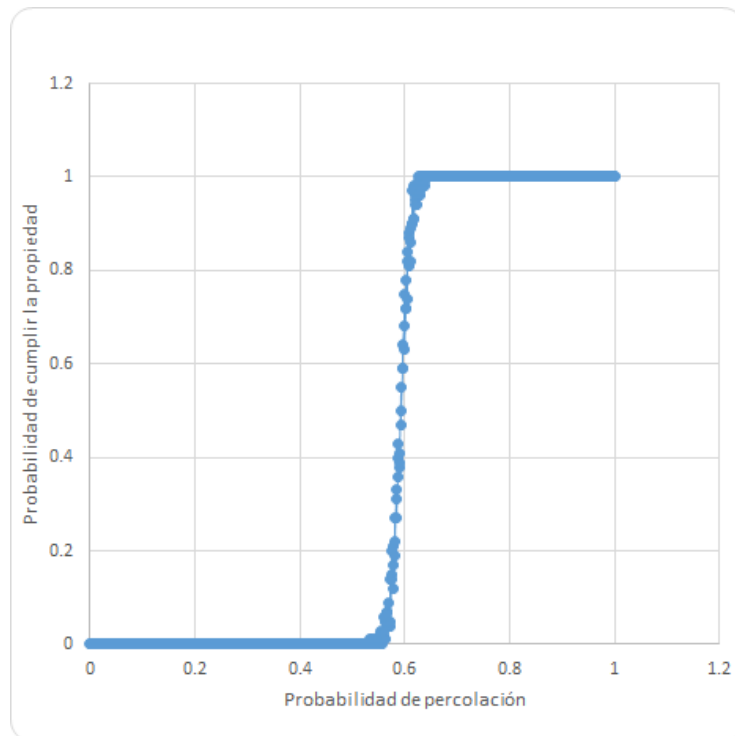


Figura 4.1.2.3 Probabilidad de cumplir la propiedad *top bottom* en función del valor de probabilidad de percolar un vértice

En esta gráfica, observamos de forma visual que el valor de transición de fase está cercano al 0.59 en la mitad de la zona donde la probabilidad empieza a subir de 0 a 1.

4.1.2.1. Coste

Empezamos con el análisis de la función *bool percolarVertices(float prob)*. Tenemos una inicialización para utilizar *union-find* de coste $O(N^2)$, a continuación tenemos un bucle que aplica una percolación de nodos de la primera fila y la última que tiene coste $O(N)$.

Por último analizamos el coste del último bucle, primero analizaremos el coste de *union-find* que asumimos para el resto de análisis como el mismo. Tenemos una implementación por *path compression* y *link by rank*. Una secuencia de m operaciones de *union* y *find* sobre n sets disjuntos tiene un coste de $O(n + m \lg n)$ que aproximando es $O(n + m)$. Por lo tanto todas las operaciones derivadas de esta estructura asumimos que tienen un coste global de $O(N^2)$, ya que como mucho se ejecutarán cuatro *union* por cada nodo seguido de la consulta de dos *find* de cada iteración.

El recorrido del último bucle en el peor de los casos será de $O(N^2 - 2N)$ ya que el recorrido no incluye la primera y la última fila que ya se ha hecho previamente.

Por lo tanto, el coste final de esta función es $O(N^2)$. Por otro lado, tenemos la función *void mainVertices(int N)*, como podemos observar va haciendo una prueba de valores que en el

peor de los casos ejecutará la función anterior 30 veces. Esto nos resultaría un coste de $O(30N^2)$ pero si lo asumimos como un factor constante nos quedaría un coste final de $O(N^2)$. Respecto a la segunda versión, es bastante más lenta ya que ejecuta el programa por cada valor de 0.001 a 0.999 100 veces, lo que son unas 100.000 veces donde hay una llamada a la función *percolarVertices* es decir sería algo parecido a $O(100.000N^2)$ donde tiene bastante más peso el factor de la izquierda que el propio N . Esto se podría optimizar reduciendo el número de llamadas a la función si reducimos la búsqueda de valores a la zona concreta donde aparece la transición de fase que en este caso es de 0.5 a 0.6 obteniendo el mismo resultado.

El vector donde guardamos la información tiene un tamaño de $N \times N + 2$, por lo tanto el algoritmo tiene un coste en espacio de $O(N^2)$.

Como añadido, destacamos que el número de vértices n del grafo es igual a $N \times N$, es decir N^2 , por lo tanto el algoritmo tiene un coste $O(n)$ sobre el número de vértices del grafo.

4.1.3. Percolación por aristas

En el caso de percolación por aristas, utilizamos la misma técnica pero introduciendo una pequeña variación. Ahora los vértices permanecen igual por lo tanto, declaramos todos como abiertos.

En el primer paso, conectamos toda la parte de arriba de la cuadrícula con el vértice ficticio porque es añadir aristas que no están en el grafo original y por lo tanto no se verán afectadas en el proceso de percolación. A partir de aquí, aplicamos el mismo concepto de *union-find* que antes y recorremos todos los vértices que restan. Para cada vértice, preguntamos a nuestra función de probabilidad si se puede abrir su arista derecha o de abajo siempre que exista. En caso de que si se abran aplicamos como mucho dos llamadas a *union*. Cada vez que se hace esta operación, como antes, se pregunta si el representante del nodo *top* es el mismo que el *bottom* en caso positivo se para la ejecución porque ya se ha cumplido la propiedad.

Siguiendo el caso de percolación por vértices, realizamos el experimento de ejecutar 100 veces con el mismo valor de probabilidad y recoger en cuántos casos se ha cumplido la propiedad para obtener una probabilidad. Realizamos lo anterior para cada valor de 0.001 en 0.001.

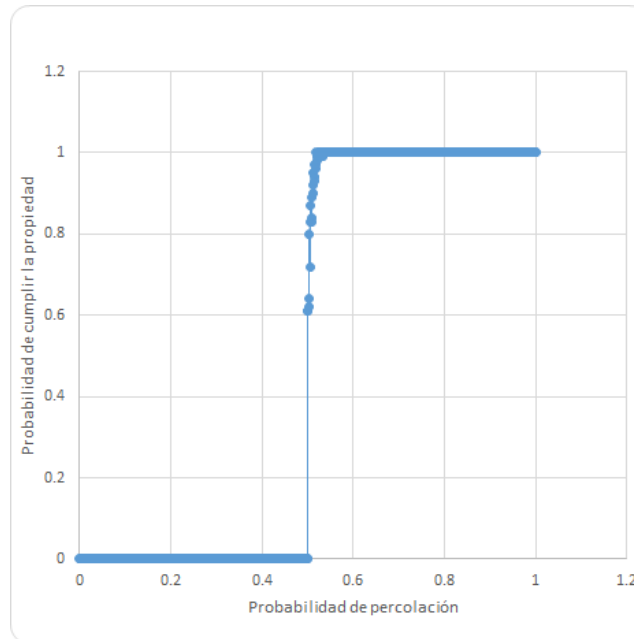


Figura 4.1.3 Probabilidad de cumplir la propiedad *top bottom* en función del valor de probabilidad de percolar una arista

Como podemos observar la zona donde hay una transición es en 0.5 y es bastante estrecha, por lo tanto nos da un valor preciso y bastante bueno.

4.1.3.1. Coste

Con respecto al coste de la función *percolarAristas*, el análisis es muy parecido a la percolación por aristas, ya que el código y el procedimiento son similares. Aún así la única diferencia es que por cada nodo como mucho se hace dos llamadas a *union* de en vez de las cuatro anteriores.

El coste final sería $O(N^2)$ en función del tamaño del lado de la cuadrícula como en el caso anterior.

4.1.4. Método MonteCarlo Vértices

Para aplicar el método de Montecarlo, primero de todo, prescindimos de la función de probabilidad que usábamos antes para decidir si un nodo se declaraba como abierto o no. Por lo tanto, ahora no tendremos explícitamente un proceso de percolación pero lo utilizaremos desde otra perspectiva diferente.

Para empezar, inicializamos todos los vértices como cerrados. Para orientar un poco y facilitar las cosas al algoritmo primero decidimos aleatoriamente qué nodos se abren de la parte de arriba de la cuadrícula y de la parte de abajo en un recorrido de coste $O(N)$.

A continuación, ejecutamos el siguiente bucle. Mientras el representante del nodo *top* no sea el mismo que del nodo *bottom*, es decir, que no estén conectados abrimos un nodo de la cuadrícula de forma aleatoria y sumamos la cantidad de abiertos a una variable. Así proseguimos hasta que conecten.

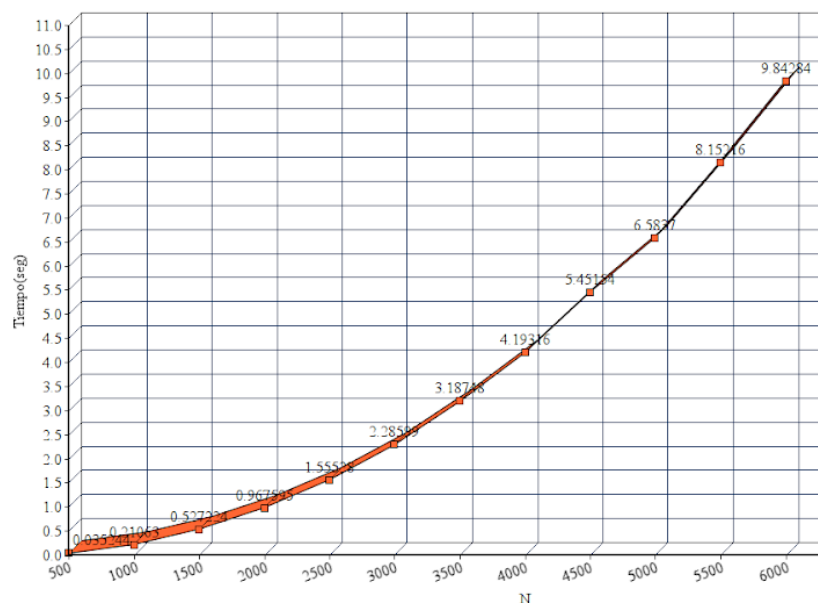
Al final, el valor aproximado para la transición de fase que estamos buscando no es más que el resultado de dividir la cantidad de nodos abiertos entre el total.

4.1.4.1. Experimento

En este experimento analizaremos los diferentes costes del algoritmo con tamaños de N creciente y los diferentes valores de la transición de fase aproximada que devuelve el algoritmo. Para realizarlo, ejecutaremos el programa desde $N=500$ sumando de 500 en 500 hasta 6000.

Para cada N , ejecutamos el programa 4 veces y recogemos la media de tiempos porque en cierto modo es bueno tomar varias medidas y coger un valor medio debido a la aleatoriedad del algoritmo.

Respecto al valor de transición de fase para cada N da un valor bastante parecido y no se puede observar una dependencia clara, por lo tanto calcularemos la media de todos los valores arrojados y ese será el resultado final.



4.1.4.1 Evolución del tiempo de ejecución en segundos en función de N

En esta gráfica, observamos cómo evoluciona el tiempo de ejecución con el tamaño de N . Claramente crece cuadráticamente tal y como habíamos explicado en el anterior análisis. Si lo observamos respecto al número vértices del grafo vemos una muy buena eficiencia ya

que con 36 millones de vértices el algoritmo encuentra una solución en menos de 10 segundos.

También cabe destacar que este algoritmo es bastante más eficiente que la anterior aproximación ya que para una N de valor 1500 devuelve el resultado en 0,5 segundos de vez de casi 3 segundos. Esto se produce debido al factor de como mucho 30 que multiplicaba a N^2 en el análisis del coste de la versión anterior de percolación por aristas que entra en juego añadiendo un coste temporal extra.

Respecto al valor del resultado, tenemos que la media de todos los valores es de **0.593119** un resultado bastante aproximado al que hemos buscado que es: 0.592746.

4.1.4.2. Coste

Sobre el coste del método *monteCarloVertices*, la inicialización del vector con un doble bucle que tiene coste $O(N^2)$ donde N es el tamaño de un lado de la cuadrícula. A continuación, como hemos mencionado previamente se decide aleatoriamente las conexiones de las partes de arriba y abajo con un bucle $O(N)$.

Por último tenemos un bucle mucho más difícil de analizar, cabe recalcar que en la generación de números aleatorios puede suceder que se decide abrir un nodo que ya está abierto, en ese caso se vuelve a escoger otro nodo. Podríamos haber cambiado la implementación para que no sucediera este caso, pero como es bastante improbable que toda la cuadrícula tenga muchos nodos abiertos sin que se cumpla la propiedad y a más ya tenemos una variabilidad en el tiempo de ejecución dependiendo de qué nodos se decidan abrir o no, consideramos que el tiempo que añaden esos casos no afecta al resultado final.

Haciendo una estimación el coste de este bucle es $O(N^2)$ en el peor de los casos. Por lo tanto el coste global de la aproximación hecha con Montecarlo es $O(N^2)$.

4.2. Top and Bottom en grafos aleatorios

En esta sección, comentaremos los pasos de implementación y experimentación que hemos realizado para estudiar la transición de fase sobre diferentes modelos de grafos aleatorios de la propiedad de top and bottom definida, inicialmente, sobre la cuadrícula. Corresponde al apartado c del enunciado y los códigos fuente programados para realizar los experimentos son *topBottomErdos.cc*, *topBottomUniform.cc* y *topBottomWatts.cc*; así como el método de testTopBottom del fichero *properties.cc*.

En los tres ficheros, para obtener los datos resultantes hemos realizado el siguiente proceso de ejecución:

- Obtener las variables que definen el modelo de grafo con el que realizaremos las pruebas, en algunos casos aleatorias o por decisión propia, en otros se reciben como entrada. También se pide si el tipo de percolación que se quiere realizar es por nodos o aristas.
- Generar los vértices que pertenecen al top y los que pertenecen al bottom. Para realizar esto, hemos decidido elegir un número determinado de vértices ($n/10$, por ejemplo) que pertenezcan a cada uno. Ningún vértice perteneciente a top puede ser adyacente con ninguno de bottom.
- El proceso de test de la propiedad. Este se basa en un conjunto de ejecución secuenciales y lo que hemos hecho es ir generando grafos de ese modelo (hemos decidido generar 100), donde cada uno se generará cuando ya hayamos testado la propiedad con el anterior, es decir, solo mantendremos un grafo en memoria en todo momento. Para cada grafo, testar la propiedad para nosotros significa probar si se cumple la propiedad con ese grafo, llamando al método *testTopBottom* con una percolación sobre el grafo que va desde una probabilidad de 0 hasta una probabilidad de 1, aumentando ésta de 0.001 en 0.001. Estos resultados parciales nos indican si la propiedad se cumple o no (1 o 0) y se almacenan en una matriz.
- Una vez realizado el test con todos los grafos, realizamos la media aritmética por tal de obtener una idea de cuál es el porcentaje en el que se cumple la propiedad.

El coste de cada experimento vendrá dado por 100 veces el coste de generar el grafo del modelo con el que estamos realizando el experimento más $100 \cdot 1000$ veces el coste de testar la propiedad top and bottom.

4.2.1. Método *testTopBottom*

El método que comprueba la propiedad se encuentra en el fichero *properties.cc*. Comprobamos la propiedad como hemos comentado en el apartado de la propiedad, pero como vamos a realizar la percolación del grafo, el método recibe, a parte de la matriz de adyacencias **por referencia constante**, el modo de percolación y la propiedad.

Dentro del método, en el caso de percolación por nodos, primero recorreremos todos los nodos y, a partir de la probabilidad de percolación, decidiremos si tener o no en cuenta el vértice. Si se decide que no, pondremos su representante a -1 para que sirva de indicativo en la determinación del cumplimiento de la propiedad.

En el caso de la percolación por aristas, el recorrido de la matriz es triangular para no visitar una arista dos veces, y, para cada arista, la probabilidad de percolación decidirá si realizar la union de los dos vértices que une o no.

El coste de la función es $O(M\log(n))$, siendo M el número de aristas y n el número de vértices, pues recorreremos la matriz de adyacencia y, para cada arista realizaremos una operación de find y, en ciertos casos, una de union, las cuales tienen coste $O(\log(n))$

4.2.2. Experimentación con Erdős–Rényi

En este experimento, analizaremos la transición de fase de la propiedad top and bottom del modelo de Erdős–Rényi. Tuvimos ciertas dudas en si aplicar o no percolación sobre grafos de este modelo, pues la generación de estos ya usa una propiedad para decidir si incluir o no una arista, pero, finalmente, decidimos hacerla por tal de generalizar el método y tener en cuenta, también, la influencia de esta probabilidad.

4.2.2.1. Percolación por vértices

Para el modelo de Erdős–Rényi, hemos querido estudiar, a parte de la transición de fase natural en grafos de este modelo, la influencia que tenía la probabilidad del propio modelo con el cumplimiento de la propiedad, como ya hemos comentado. Para hacerlo, hemos realizado el estudio de transición de fase estándar con 3 valores distintos en la probabilidad de generación de las aristas del grafo.

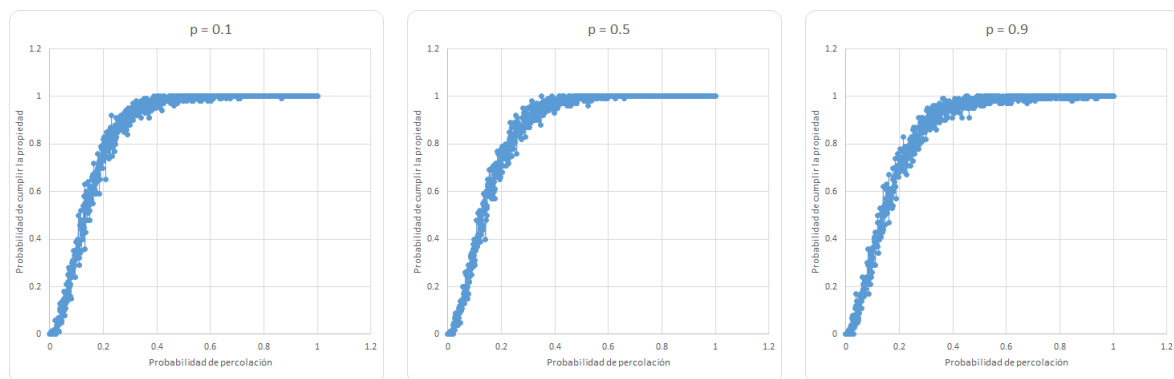


Figura 4.2.1. Probabilidad de percolación para probabilidades de 0.1, 0.5 y 0.9

Como se puede observar en las gráficas de la figura 4.2.1, la influencia de la probabilidad que recibe el modelo para generar el grafo no influye de manera notable. Considerando 0.8 una probabilidad considerable de que se cumpla la propiedad, en los tres casos el resultado giraría entorno a 0.2. Estos resultados se ven claramente influenciados por la decisión de top y de bottom realizadas, que, en nuestro caso, para este modelo simplemente selecciona los primeros vértices adyacentes entre ellos como top y los primeros que no son adyacentes a top, como bottom. Una variación que habría sido interesante estudiar es la de ir cambiando el criterio de selección de éstos y reducir su tamaño.

4.2.2.2. Percolación por aristas

Para el proceso de percolación por aristas hemos seguido la misma metodología, pero, en este caso, los resultados han sido relativamente distintos:

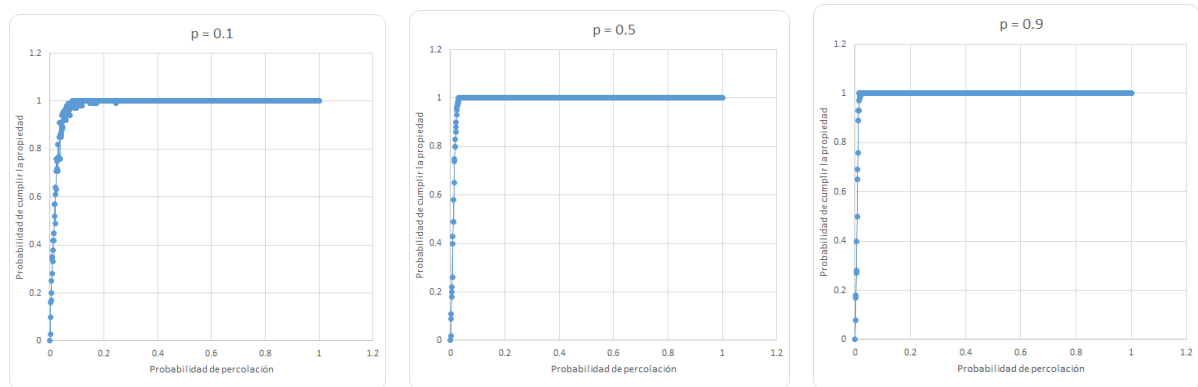


Figura 4.2.2. Probabilidad de percolación para probabilidades de 0.1, 0.5 y 0.9

Como se puede observar en las gráficas de la figura 4.2.2, los resultados en este caso cambian de forma notoria. De entrada, el cumplimiento de la propiedad es mucho mayor, a partir de entorno a 0.05 o incluso antes, y se mantiene estable una vez se consigue una probabilidad certera, de 1.

Por otro lado, esta vez las diferencias son más visibles alterando la probabilidad de Erdős–Rényi. Los resultados obtenidos son, si más no, lógicos, pues a menor probabilidad, más cuesta conseguir que se cumpla la propiedad. Aún así, el valor de la transición de fase sigue siendo muy pronto.

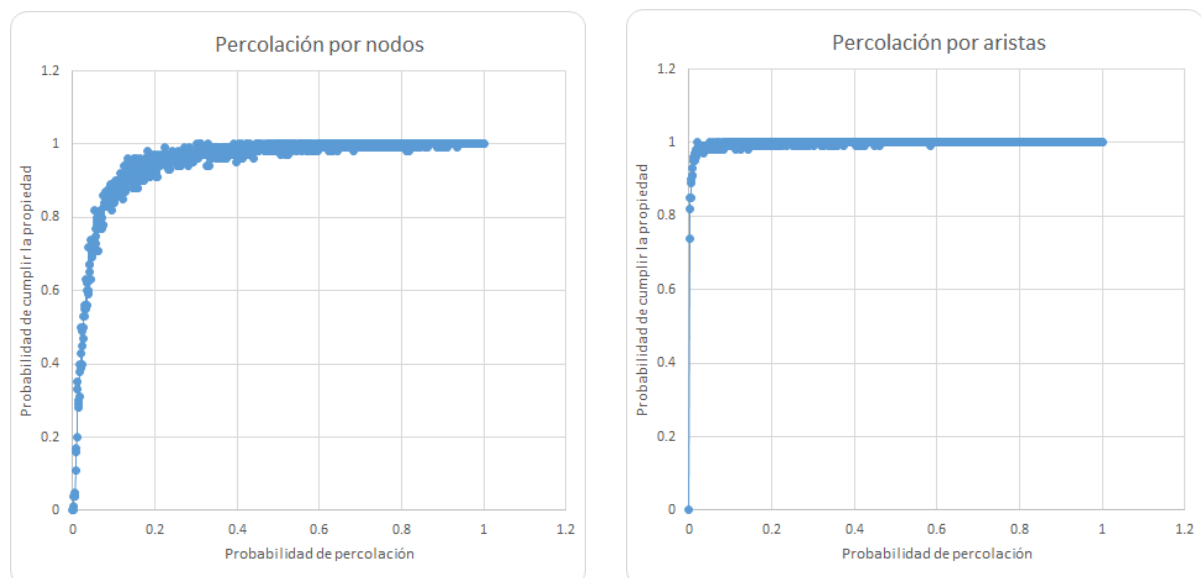
4.2.2.3. Conclusiones

Como conclusión de esta experimentación podemos afirmar que el modelo de Erdős–Rényi genera grafos que cumplen fácilmente la propiedad, pues, incluso con probabilidad 0.1 conseguimos que se cumpla la propiedad en un valor bastante bajo de percolación. Por otro lado, creemos que la elección de top y bottom tiene bastante que ver con los resultados obtenidos. Una buena variación sería cambiar este criterio.

4.2.2. Experimentación con el modelo uniforme

En este experimento, analizaremos la transición de fase de la propiedad top and bottom del modelo uniforme. Para ello, la metodología aplicada es la misma que en el modelo de Erdős–Rényi, hemos realizado los tests para diferentes valores de percolación y anotado la probabilidad con la que se cumple la propiedad probando 100 grafos generados siguiendo este modelo. Algo a destacar que hemos notado es que el tiempo de ejecución de este experimento era notablemente superior. Esto es debido a que el tamaño de los grafos es variable (desde 50 hasta 6000 vértices), con lo cual cuando hay que estudiar grafos de mayor tamaño el tiempo de ejecución, lógicamente, es mayor.

Los resultados obtenidos los hemos representado en las gráficas adjuntadas a continuación (figura 4.2.3). Como es lógico, vemos que la transición de fase se sitúa en una probabilidad de percolación mayor para nodos que para aristas. Es lógico pues al invalidar un nodo se están descartando un conjunto de aristas (todas las que son incidentes a éste) y, en el proceso de percolación por aristas solamente descartamos aristas individualmente.



4.2.3. Resultados obtenidos al percolar grafos del modelo uniforme

Como se puede observar, el valor de transición de fase en el proceso de percolación por nodos se sitúa entorno a 0.1, mientras que en el proceso de percolación por aristas se sitúa cercano al 0. El 0, obviamente, no puede hacer que se cumpla la propiedad, pues no permites que se añada ninguna arista.

4.2.3. Experimentación con el modelo de Watts-Strogatz

En este experimento, analizaremos la transición de fase de la propiedad top and bottom del modelo de Watts-Strogatz. Para ello, la metodología aplicada en este caso es ligeramente distinta a las dos anteriores. La diferencia recae en la elección de top y bottom. Mientras que en los dos modelos anteriores no tenemos conocimiento profundo de la estructura que sigue el grafo, en el modelo de Watts-Strogatz sí podemos hacernos una idea de la forma que puede tener. Nuestra forma de elegirlo, para este modelo, consiste en seleccionar un grupo de vértices vecinos como top, y un grupo de vértices que son los más alejados como bottom. En la figura 4.2.4 se ilustra un ejemplo partiendo de la figura 2.2.

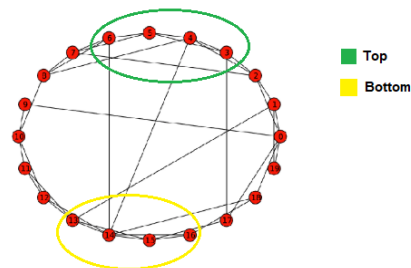


Figura 4.2.4 Ejemplo de selección de top y bottom

Hemos tomado esa decisión por tal de intentar que la propiedad, aplicar percolación, fuera más difícil que se cumpliese.

4.2.3.1. Percolación por nodos

Para analizar si nuestra selección ha tenido influencia sobre el resultado hemos realizado el test con los dos criterios de selección de top y bottom.

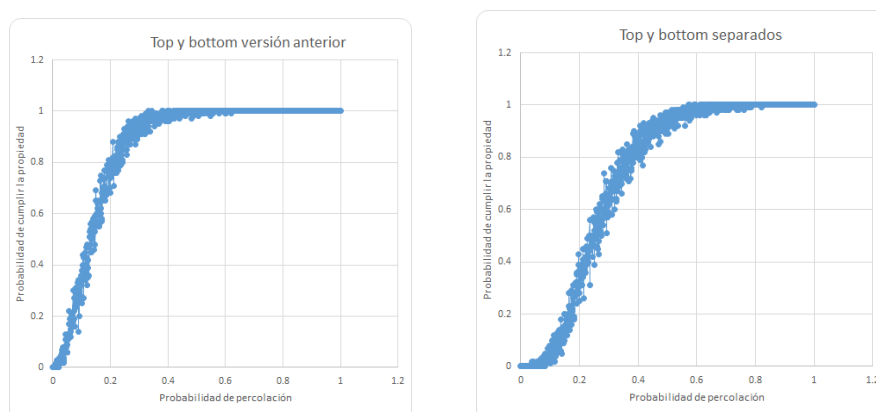


Figura 4.2.5. Probabilidad de percolación con las dos selecciones de top y bottom

Como podemos observar en la figura 4.2.5, los resultados han sido los esperados. Al seleccionar un top y un bottom más separados (los más separados, de hecho), el cumplimiento de la propiedad ha resultado más complicado, encontrando la transición de fase para esta última versión en un valor entorno a 0.4, frente a un valor entre 0.2 y 0.3 para la versión usada en los apartados anteriores. Destacar también que, en comparación a los otros dos modelos ya comentados, para este modelo al aplicar percolación es más difícil que se cumpla la propiedad.

4.2.3.2. Percolación por aristas

Para el proceso de percolación por aristas hemos realizado la misma comparativa y, como podemos observar en la siguiente gráfica de la figura 4.2.6, los resultados obtenidos son parecidos.

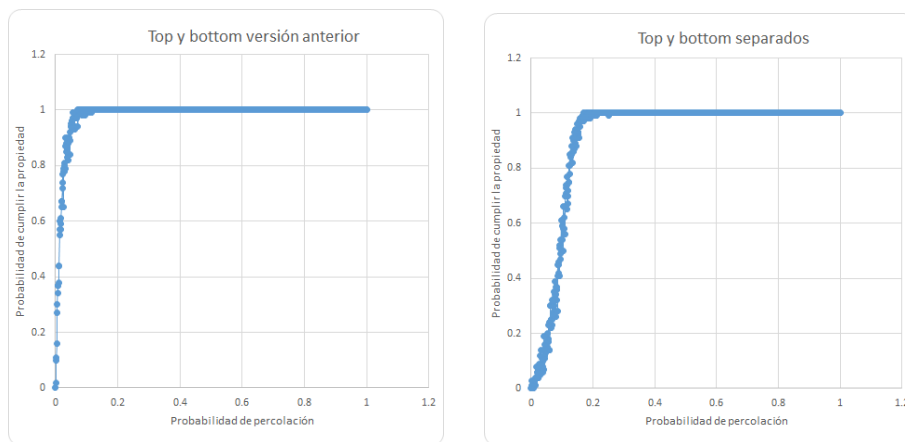


Figura 4.2.6. Probabilidad de percolación con las dos selecciones de top y bottom

Vemos que, al igual que en el proceso de percolación por nodos, la selección de top y bottom ha tenido un efecto en los resultados, aunque en este caso menos notables, ya que la propiedad se cumple antes en el proceso de percolación por aristas.

En este caso, la propiedad se cumple para valores entre 0.1 y 0.2 para la selección nueva y cerca de 0.05 para la selección ya usada.

4.2.3.3. Conclusiones

Como conclusión de esta experimentación podemos decir que la selección de top y bottom, en este modelo, tiene sentido y efecto en los resultados. Por otro lado, el resultado de la transición de fase ha sido como esperábamos, pues siguiendo la estructura de referencia que presenta este modelo, esperábamos que la transición de fase se situará en valores superiores a los encontrados en los dos experimentos anteriores y así ha sido.

4.3. Grafos cíclicos

El objetivo de estos experimentos es ver cómo afecta aplicar un proceso de percolación sobre grafos con ciclos, es decir, estudiar la transición de fase para la que estos grafos pasan de ser acíclicos a cíclicos.

El proceso de experimentación para estudiar esta propiedad se ha realizado de la misma forma que la propiedad anterior. Hemos generado 100 grafos y para cada uno hemos testado si se cumple la propiedad o no con los diferentes valor de percolación de 0 a 1.

4.3.1. Método *testCycle*

El método *testCycle* que hemos implementado hace uso de la estructura de datos Union-Find. Para ver en detalle la implementación ver el apartado 3.4. En el método, al igual que en el de *testTopBottom*, hemos realizado la percolación al momento de comprobar la propiedad, exactamente de la misma forma que en el experimento anterior. Para realizar el proceso de percolación por nodos, invalidamos los vértices que se borran poniendo el representante a -1, y en la búsqueda esos vértices no los tenemos en cuenta. Para la percolación por aristas, al ir recorriendo la matriz de adyacencias decidimos si contar o no una arista. El coste del algoritmo es, teniendo en cuenta el que ya habíamos calculado del algoritmo sin la percolación, que era $O(M \log(n))$, ahora, para cada arista, es decir, M veces, haremos una operación de find y, en algún caso, una de union, por tanto, por cada arista hacemos un número constante de operaciones de coste $O(\log(n))$, ergo el coste del algoritmo con la inclusión de la percolación se mantiene y es $O(M \log(n))$.

4.3.2. Experimentación en modelo cíclico

Para el modelo cíclico, nos hemos limitado a estudiar la transición de fase en un proceso de percolación por vértices y por aristas por tal de compararlas. El modelo de grafos que hemos llamado cíclico genera grafos con un ciclo o más, por tanto resulta ideal para la propiedad que queremos comprobar

4.3.2.1. Percolación por vértices

Como podemos ver en la figura 4.3.1, con el método de percolación por aristas conseguimos una transición de fase entorno a 0.2. Cabe destacar, por eso, la variación en los resultados, encontrándonos casos de fallo incluso en valores cercanos al 1.

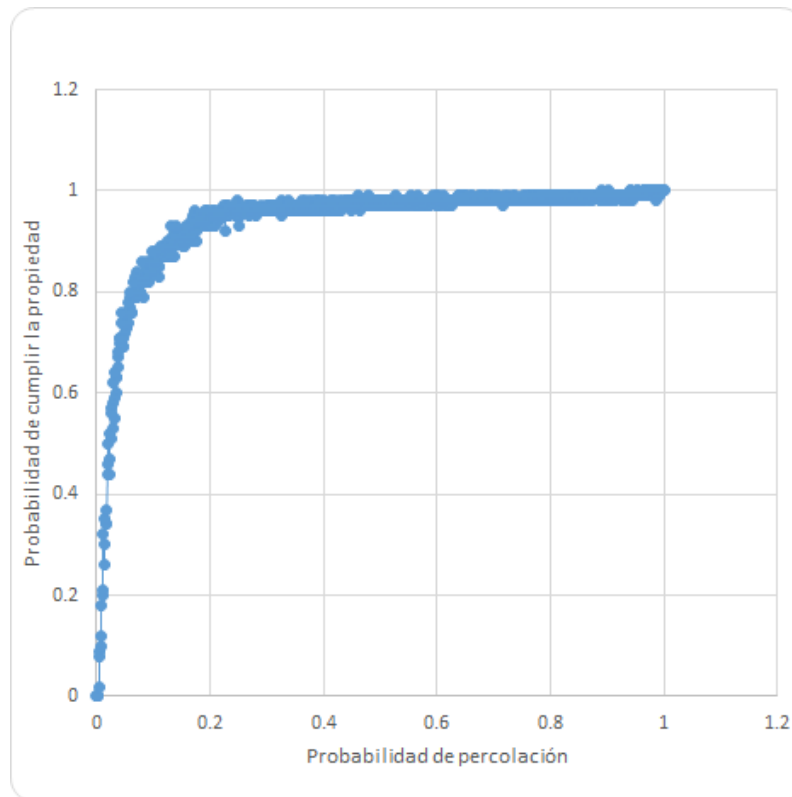


Figura 4.3.1. Resultados obtenidos al percolar por nodos

4.3.2.2. Percolación por aristas

En el proceso de percolación por aristas, como pasaba en la propiedad de test and bottom, tenemos que se cumple más rápidamente la propiedad que en un proceso de percolacion por aristas.

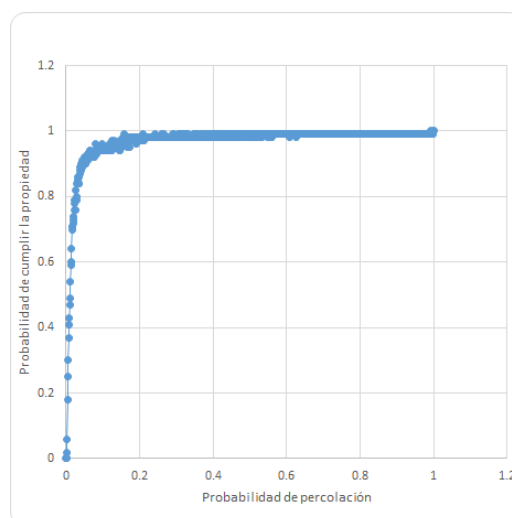


Figura 4.3.2. Resultados obtenidos al percolar por aristas

Como se puede ver en la figura 4.3.2, la transición de fase se encuentra entorno a 0.1.

4.3.2.4. Conclusiones

Para este experimento las conclusiones extraídas son que, con el modelo generado se podría explotar más la experimentación variando el número de ciclos que tiene el grafo, en lugar de hacerlo aleatorio, y seguimos corroborando la idea de que el proceso de percolación por nodos es más destructivo que por aristas, como es de suponer.

4.3.2. Experimentación en modelo de Watts-Strogatz

Para el experimento sobre el modelo de Watts-Strogatz hemos pensado una test alternativo para analizar. El método de creación de grafos de este modelo, recibe, entre otros, el valor k , que nos indica el número de vecinos a los que cada nodo será adyacente. Hemos pensado que eso determina claramente el número de ciclos que tendrá el grafo y, por lo tanto, al aumentar ese valor, la transición de fase disminuirá. Para hacerlo, hemos decidido hacer el test habitual sobre 3 valores de k : 1, 5 y 20, bajo los dos procesos de percolación, el de aristas y el de vértices.

4.3.2.1. Percolación por vértices

Para este experimento, los resultados que esperamos son de una transición de fase alta para el $k=1$, que ésta decrezca notablemente para $k=5$ y más aún para $k=20$.

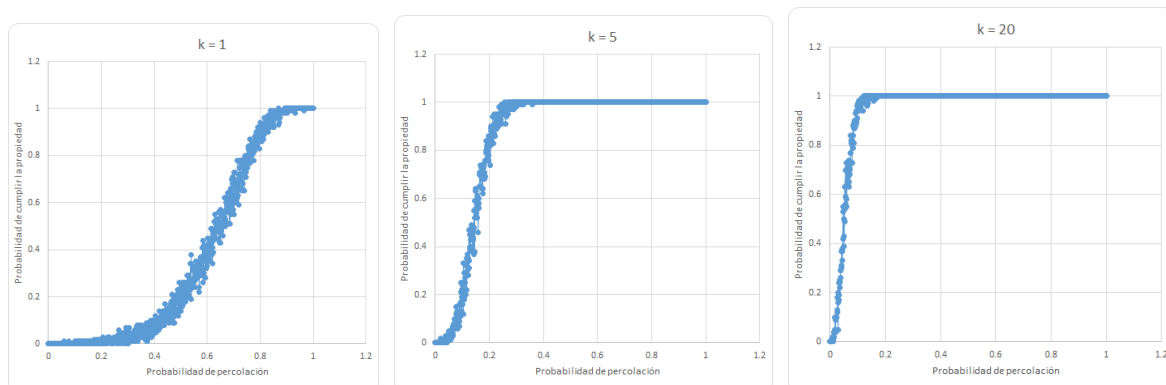


Figura 4.3.3. Resultados obtenidos de percolación con diferentes valores de k

Como vemos en la figura 4.3.3, los resultados esperados se ven reflejados en la realidad. Con $k=1$ La transición de fase se acerca a la probabilidad de percolación 1, para $k=5$ esta entorno a 0.2 y para $k=20$ entorno a 0.1.

4.3.2.2. Percolación por aristas

Para el proceso de percolación por aristas los resultados son similares, aunque con una más rápida transición de fase.

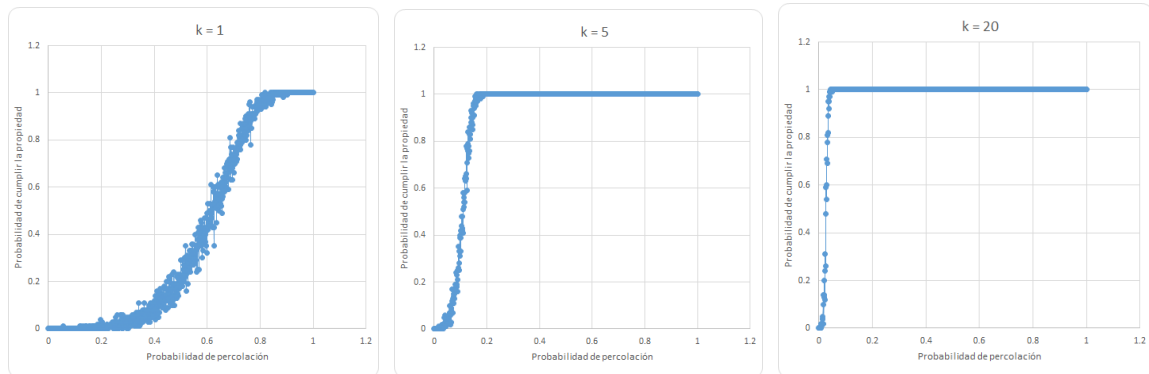


Figura 4.3.3. Resultados obtenidos de percolación con diferentes valores de k

Como vemos en la figura 4.3.3, los resultados obtenidos coinciden con lo que pensábamos antes de realizar el experimento, aquí también vemos que la transición de fase se ve afectada por el número de vecinos que determinamos, estando así para $k=1$ localizada entorno a 0.8, para $k=5$ entorno a 0.2 y para $k=20$ entorno a 0.05.

4.3.2.3. Conclusiones

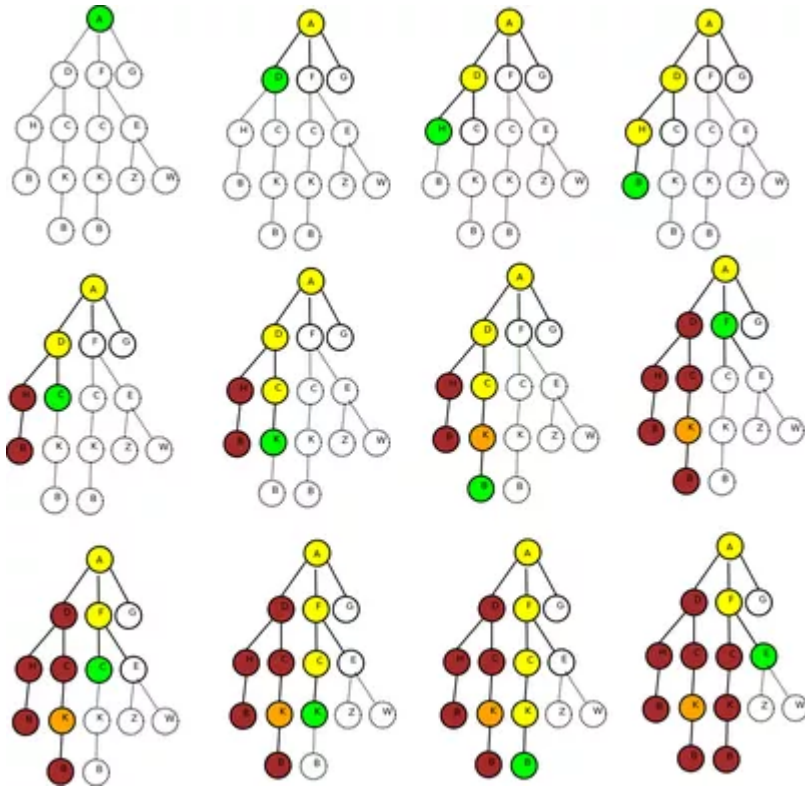
Como conclusión de este experimento podemos afirmar que la influencia del grafo que generamos puede llegar a tener más peso en la transición de fase que el proceso de percolación que apliquemos. Hemos observado la gran influencia que tiene el número de vecinos que decidamos en el modelo de Watts-Strogatz sobre el cumplimiento de la propiedad que analizamos.

4.4. Grafos conexos

En este apartado explicaremos cómo utilizamos diferentes modelos de generación aleatoria de grafos para encontrar transiciones de fase sobre la propiedad de un grafo de ser conexo.

4.4.1. Búsqueda en profundidad (DFS)

El algoritmo utilizado para comprobar que un grafo es conexo es el algoritmo de búsqueda en profundidad, este algoritmo nos sirve para recorrer todos los nodos de una componente conexa de manera ordenada, expandiendo todos los nodos que va localizando.



Funcionamiento de la búsqueda en profundidad en un grafo

El coste del algoritmo de búsqueda en profundidad es $O(n + m)$, donde n es el número de vértices y m el número de aristas.

4.4.2. Experimentación usando el modelo de Erdos

En este apartado hemos mirado donde ocurre la transición de fase en base a la probabilidad p del modelo de Erdos de generar una arista o no. El resultado para un grafo con 100 nodos fue que la transición de fase aparece alrededor del 0.09.

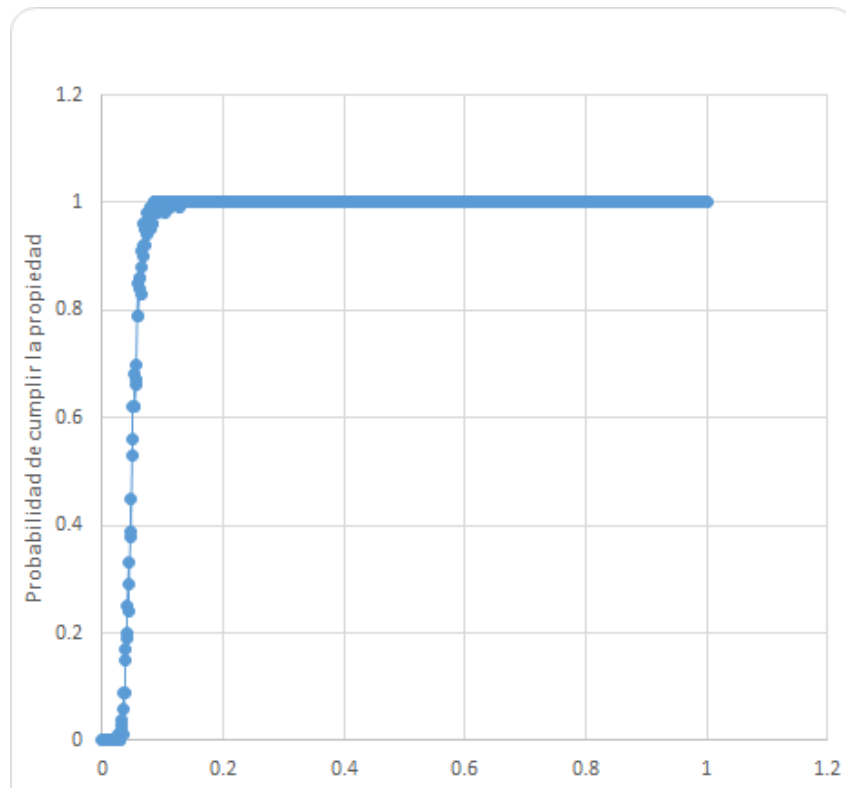


Figura 4.4.2: Probabilidad de ser conexo en el modelo de Erdos respecto a p

Al realizar este experimento nos dimos cuenta de que el lugar donde aparece la transición de fase depende directamente del tamaño del grafo, debido a que para cada nodo la probabilidad de que ese nodo conecte con al menos otro es la opuesta de que no conecte con ninguno que es $(1-p)^{(n-1)}$, es decir disminuye cuantos más nodos haya en el grafo, por lo que la probabilidad de que conecte con otro aumenta, esto nos sirve para darnos cuenta intuitivamente de porque la transición de fase en este caso depende del tamaño.

4.4.3. Experimentación usando el modelo uniforme

En este apartado hemos mirado donde ocurre la transición de fase en base al número de aristas de un grafo. El resultado para un grafo con 100 nodos fue que la transición de fase aparece a partir de la arista número 400.

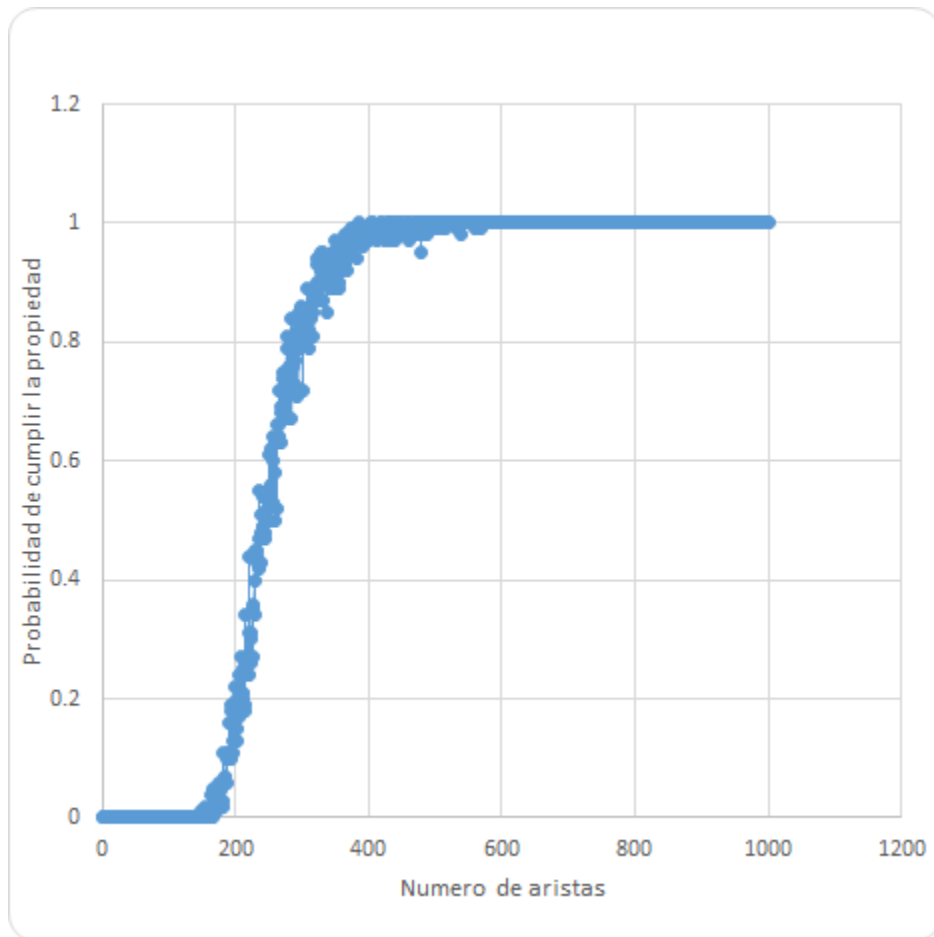


Figura 4.4.3: Probabilidad de un grafo con 100 nodos respecto al número de aristas

En este caso la transición de fase aparece antes de lo esperado, nos sorprendió que pese a que el número máximo de aristas que se pueden poner en un grafo sin que este sea conexo son $\frac{(n-1)*(n-2)}{2}$, que para el caso son más de 4000 aristas, al colocar tan solo 400 aleatoriamente ya conseguimos que fuese conexo.

4.5 Grafos Eulerianos

En este apartado explicaremos cómo utilizamos diferentes modelos de generación aleatoria de grafos para encontrar transiciones de fase sobre la propiedad de contener un ciclo Euleriano.

4.4.1 Método de comprobación

Recordemos que para ver que un grafo es Euleriano basta con comprobar que es conexo (tras remover los vértices aislados) y que todos sus vértices tienen grado par. Para comprobar estas dos condiciones lo que hicimos fue utilizar el método de búsqueda en profundidad utilizado anteriormente, aprovechando el recorrido para comprobar que los grados de los vértices son pares.

4.4.2 Ciclo Euleriano

Después de comprobar la generación con diferentes modelos aleatorios vimos que la propiedad de ser Euleriano no mostraba una transición de fase ya que generando grafos de manera aleatoria muy difícilmente cumple la condición:

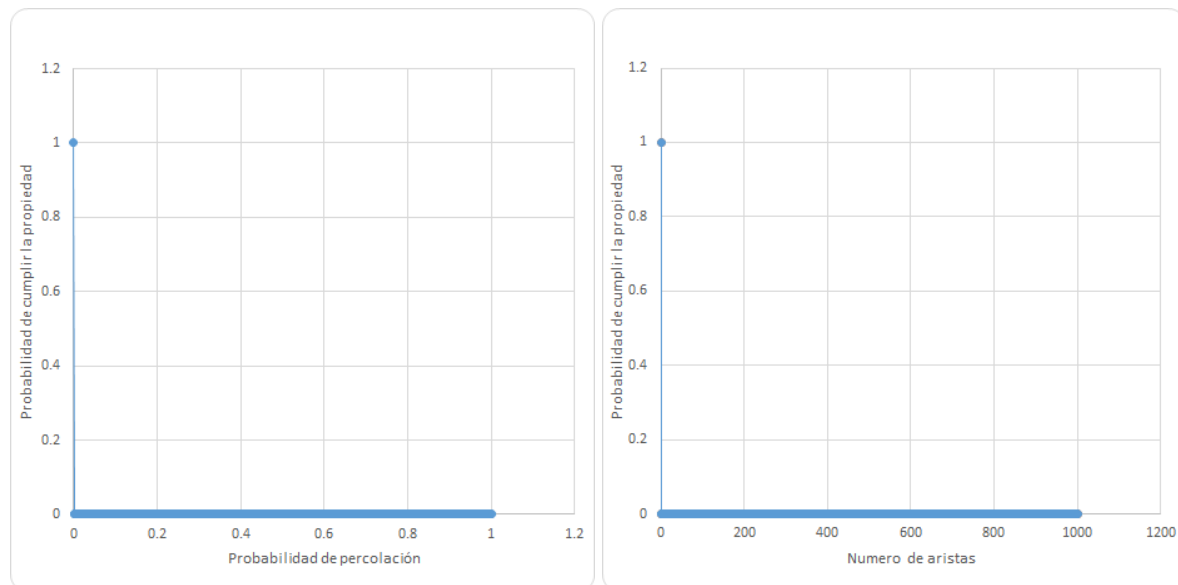


Figura 4.4.2: Gráficas para la propiedad de ser Euleriano

Tras ver los resultados llegamos a la conclusión de que generando un grafo de manera aleatoria era muy complejo que éste contuviese un ciclo Euleriano, nuestro razonamiento fue el siguiente:

Suponiendo que un grafo aleatorio cumpliera la primera condición de solo tener una componente conexa quitándole los vértices aislados, la probabilidad de que el grado de un vértice sea par es 0.5, independientemente de la manera de generar las aristas, por lo que la probabilidad de que el grado de todas las aristas sea par es $0.5^{n/2}$ siendo n el número de nodos conectados, muy complejo a la que n crece.

4.4.3 Camino Euleriano

Tras ver el poco éxito a la hora de generar grafos Eulerianos aleatoriamente intentamos mirar si con una propiedad un poco más laxa como es contener un camino euleriano la probabilidad de éxito fuese más alta.

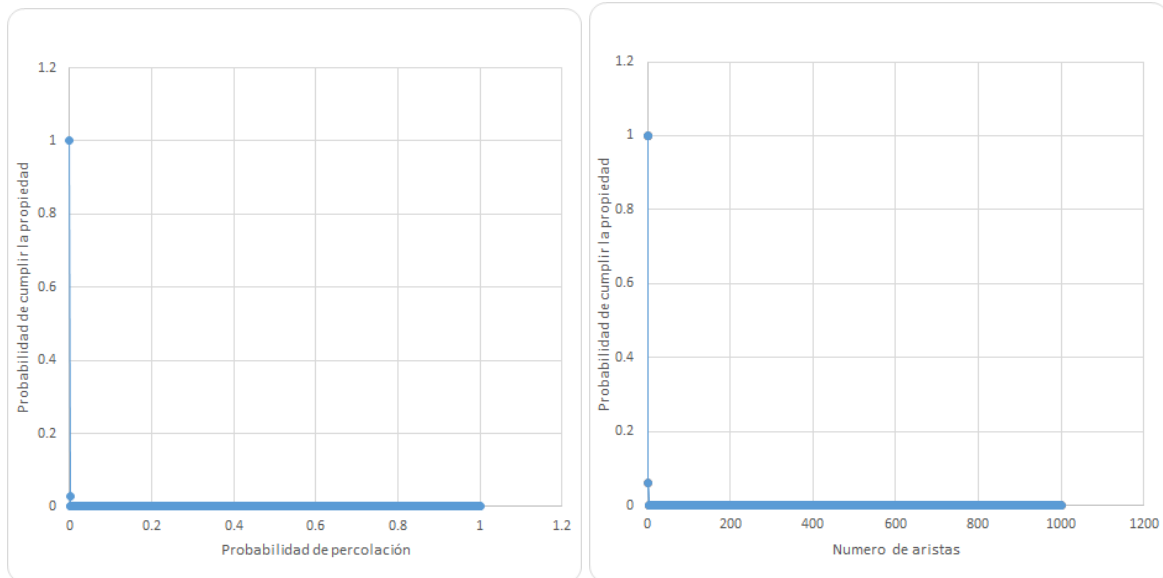


Figura 4.4.3: Gráficas para la propiedad de contener un camino Euleriano

Como podemos comprobar en los resultado no creció la probabilidad de éxito ya que la condición para contener un camino euleriano en un grafo es sólo vagamente más laxa que la de contener un ciclo Euleriano, en concreto $0.5^{n/2-1}$.

5. Conclusiones

Este proyecto nos deja diferentes conclusiones que se resumen a continuación:

En primer lugar en cuanto a organización este proyecto nos ha mostrado por encima de cualquier otra cosa es lo importante que es tener un plan trazado antes de empezar a trabajar, habiendo comprendido que la pauta a seguir del enunciado del proyecto era escueta a propósito debimos haber hablado sobre qué rumbo íbamos a tomar delante de dudas que iban a surgir y que, de forma individual resolvimos con diferentes decisiones, como se puede observar en los diferentes apartados de este documento. Esto, por otra parte, pese a que nos ha llevado a tener que realizar un mayor esfuerzo en homogeneizar el proyecto nos ha aportado mayor riqueza lectiva ya que hemos explorado un mayor número de posibilidades y hemos tenido que discutir y argumentar sobre ello entre nosotros.

En segundo lugar en cuanto a conocimiento nos ha servido para estudiar y repasar diferentes conceptos sobre grafos además de aprender algunos conceptos sobre ellos que no conocíamos. Específicamente en el campo de los modelos de grafo aleatorio, uno nuevo para nosotros, creemos que la realización de esta práctica nos ha servido como una buena introducción a este tema.

Finalmente a nivel práctico esta práctica ha sido útil para investigar sobre la implementación de algunos algoritmos o estructuras vistas en clase e implementar alguna de ellas. También ha servido para mejorar nuestra capacidad de obtener y analizar datos desde nuestro código.

6. Bibliografia

- [1] “Erdős–Rényi model,” *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Erdős–Rényi_model [Accessed: 10-May-2019].
- [2] A. de Mier and M. Maureso, “Teoria de grafs.” Departament de matemàtiques. FIB., 2019.[Accessed: 10-May-2019].
- [3] “Small-world network,” *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Small-world_network [Accessed: 08-May-2019].
- [4] “Watts–Strogatz model,” *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Watts–Strogatz_model [Accessed: 07-May-2019].
- [5] E. W. Weisstein, “Eulerian Graph,” *Mathworld*. [Online]. Available: <http://mathworld.wolfram.com/EulerianGraph.html> [Accessed: 08-May-2019].
- [6] “Graph generators,” *NetworkX Developers*, 2017. [Online]. Available: <https://networkx.github.io/documentation/networkx-2.0/reference/generators.html> [Accessed: 6-May-2019].
- [7] “Source code for networkx.generators.random_graphs,” *NetworkX Developers*, 2017. [Online]. Available: https://networkx.github.io/documentation/networkx-2.0/_modules/networkx/generators/random_graphs.html#newman_watts_strogatz_graph [Accessed: 9-May-2019].
- [8] “Random graph,” *Wikipedia*, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Random_graph [Accessed: 6-May-2019].
- [9] “Monte Carlo method,” *Wikipedia*, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Monte_Carlo_method [Accessed: 7-May-2019].
- [10] “Case study: Percolation,” *Princeton University*, 2019. [Online]. Available: <https://introcs.cs.princeton.edu/python/24percolation> [Accessed: 8-May-2019].
- [11] “Percolation on a Square Grid,” *Wolfram Demonstration Project*, 2019. [Online]. Available: <http://demonstrations.wolfram.com/PercolationOnASquareGrid> [Accessed: 9-May-2019].
- [12] “Union-Find” *Princeton University*, 2019. [Online]. Available: <https://algs.cs.princeton.edu/lectures/15UnionFind-2x2.pdf> [Accessed: 3-May-2019].

