Percolación de sitio y probabilidad crítica

Universidad Nacional de Colombia
Facultad de Ciencias, Departamento de Física
Herramientas Computacionales
Profesor: William Fernando Oquendo Patiño
C. Angel Q., M. Garcia R.
cfangelq@unal.edu.co, miagarciaru@unal.edu.co

17 de marzo de 2020

Departamento de Física, Universidad Nacional de Colombia, Bogotá, Colombia.

Resumen

El siguiente proyecto tiene como fin estudiar la percolación de sitio para una red cuadrada de dimensión L. El llenado de ésta red está sujeto a una probabilidad p. Para esto, se realizan simulaciones variando tanto la probabilidad de llenado p como las dimensiones de la red para determinar si hay formación de un cluster percolante para cada p y L escogidos. Éste procedimiento se repite un total de 20 veces para cada simulación con valores de p y L fijos. Variando las probabilidades de llenado partiendo desde 0 hasta alcanzar 1, junto con las dimensiones de la red, es posible determinar una probabilidad crítica de llenado para la cual el sistema tiene percolación o no (probabilidad p_c). Adicionalmente, se calcula el tamaño del clúster percolante más grande para todas las simulaciones realizadas, siendo expresada esta cantidad en un resultado normalizado sobre las dimensiones de la red LxL para cada probabilidad y dimensión ($s(p, L)$). Finalmente se estudia el tiempo empleado por el programa cuando la red aumenta en tamaño para dos niveles de optimización del compilador, junto con el registro de profiling indicando el flat profile tomado para $L = 128$ y $p = p_c \approx 0,59271$.

Palabras clave: Percolación de sitio, cluster, probablidad crítica, matriz aleatoria.

Introducción

Aquellas casillas que conforman la red se llenarán con una probabilidad p (probabilidad de llenado p entre $[0, 1]$) de manera que cuando la simulación del sistema haya terminado, se formarán aglomeraciones de casillas que fueron llenadas o están vacías.

Estos grupos de casillas llenas pueden formar clusters de distintos tamaños dependiendo de la probabilidad de llenado p y las dimensiones de la red L. Un cluster se conforma de casillas que están llenas y además son vecinas entre ellas, es decir: hacia arriba, abajo, a derecha y a izquierda (no se consideran en dirección diagonal). Observe la figura (1), en la cual se han representado clusters de un llenado con una probabilidad p entre $[0, 1]$ y $L = 19$. La separación de los clusters se da en las diferentes tonalidades de color que se muestran. Así, por ejemplo, puede observarse que para casillas aledañas (arriba, abajo, derecha e izquierda) son consideradas de un mismo cluster, mientras que aquellas casillas en diagonal que también están llenas no hacen parte del mismo cluster.

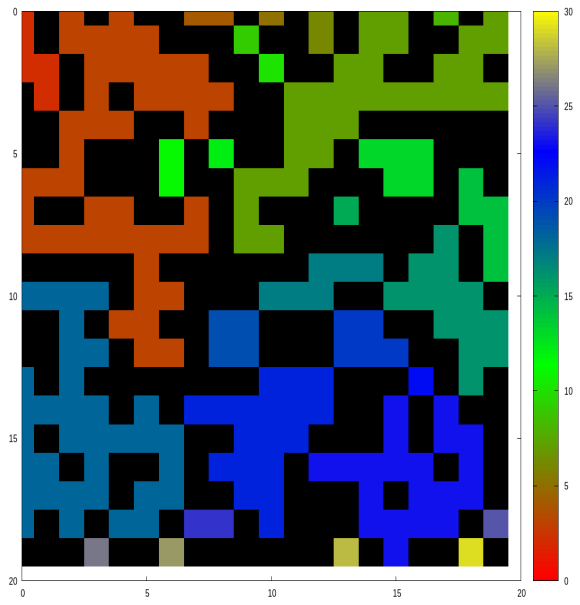


Figura 1: Clusters para un llenado de aleatorio de una red 19x19.

Si existe un clúster que sea capaz de unir a través de una cadena de casillas a una de estas situada en la fila superior a otra de la fila inferior de la red, o de izquierda a derecha, se dice que el sistema percola o es percolante. Al clúster que cumple con tales condiciones se le denomina clúster percolante.

En la figura (1) no es posible observar la existencia de un clúster percolante, pues observe que no existe una cadena de casillas llenas (casillas del mismo color) que haya alcanzando a unir la fila superior con la fila inferior, o la primera columna con la última, por lo que en éste caso el sistema no es percolante. Las casillas negras son aquellas que están vacías.

A medida que aumentan la probabilidad de llenado, también aumentan el número de casillas llenadas, y por ende la cantidad de clusters junto con la probabilidad de que el sistema percole. Por otra parte, la probabilidad de que el sistema percole $P(p, L)$ también depende de la dimensión de la red, por lo que estudiaremos su efecto sobre el número de clústers, el tamaño de cada clúster y la probabilidad de percolación P.

Variando la probabilidad de llenado p para un mismo sistema es posible encontrar una probabilidad crítica p_c para la cual el sistema genera un cluster percolante. Por lo tanto, ésta probabilidad p_c es la mínima probabilidad de llenado para la cual el sistema percola.

Metodología

Con el fin de generar las matrices de forma pseudo-aleatoria se empleo el generador "Mersenne Twister"(identificado en C++, en la libería random, como mt19937) sobre una distribución uniforme. Este algoritmo sobrepasa los requerimientos del problema puesto que el tamaño máximo de matriz que se usará es de 512 – esto implica un máximo de 262144 distintos elementos –, mientras que el periodo del generador es de $(2^{19937} - 1)$ [1]. Al suponer que los números se generan de forma aleatoria y distribuirlos sobre un rango [0,1], se espera que la probabilidad de que aparezca un numero sea independiente de su valor. Al interpretar rangos de valores de los números como percentiles y suponer que, según ello, las casillas de las matrices se encuentran ≪abiertas≫ o ≪cerradas≫ se espera tener una matriz con una probabilidad de llenado: por ejemplo, si se toma una probabilidad de 0.3 de que una casilla se encuentre abierta, se asume que las entradas de la matriz para los cuales los datos tienen valor $< 0{,}3$ se encuentran abiertas, y se les asigna un valor de 1, el resto (entradas con valor $> 0{,}3$) se encuentran cerradas.

Para encontrar los distintos clústers que se forman y la percolación de estos se empleó un algorítmo tipo DFS (depth-first search) , dado que se busca analizar todos los posibles clústers, de su nivel en terminos de ramificación [2]. Adaptado al problema, para ello se genera un vector o array de tipo bool que hace de índice, basándose en las casillas abiertas y cerradas de la matriz pseudo-aleatoria. Seguido de esto se inicia un barrido de este array. Al encontrar una casilla abierta se inicia un algoritmo-explorador el cual chequea si las casillas adyacentes (no-diagonales) se encuentran abiertas, se marcan con id y su posición es añadida a un buffer, el algorítmo empieza a correr iniciando ahora sobre uno de los valores que se encuentran en el buffer, el valor sobre el cual se inicia algoritmo ahora es eliminado del buffer. Al no encontrar más casillas adyacentes se continua el barrido del array y al encontar un nuevo clúster sus casillas son marcadas con otro id. El tamaño del clúster se conoce al tener un contador que aumenta cada vez que el algoritmo corre.

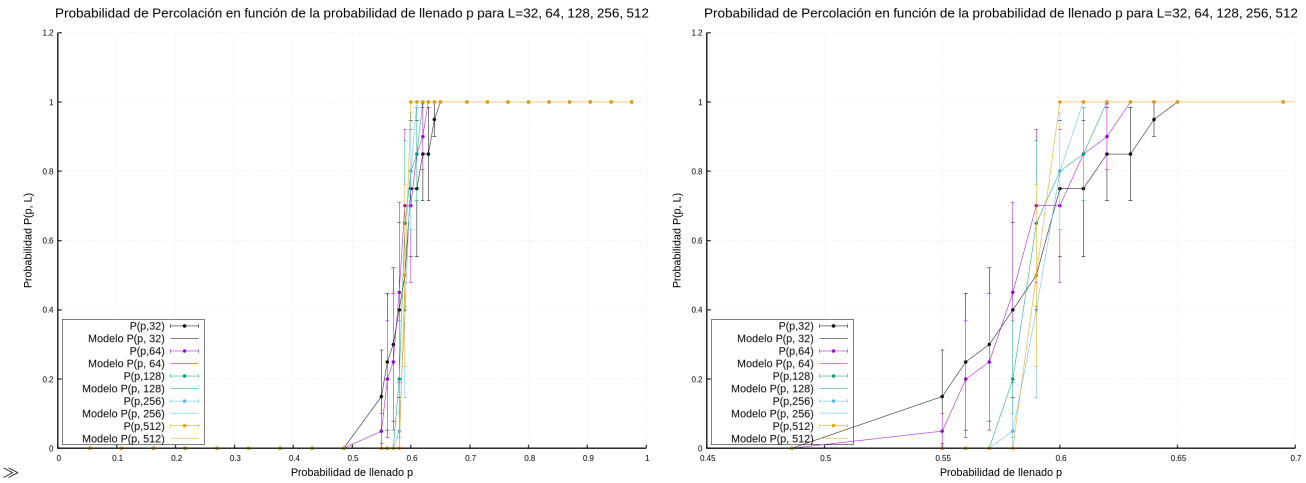
El proceso se realizo para matrices de tamaño $L = 32, 64, 128, 256, 512$, 30 valores distintos de probabilidades entre 0 y 1. Para cada combinación de tamaño y probabilidad se generaron 20 matrices distintas valiéndose del generador, esto, en teoría, mantiene las propiedades generales para los cluster (ver figura 5). Con el fin de asegurar reproducibilidad se tomaron valores conocidos de semilla en el generador, 0 a 20. Se sacaron estadísticas del tamaño del clúster percolante más grande en función de la probabilidad de llenado, normalizadas al tamaño de la matriz, así como la desviación estándar de estos valores. Por otro lado, se sacó la probabilidad de que existiera un cluster percolante, asi como su desviación, en función de la probabilidad de llenado y tamaño de matriz.

El programa fue compilado con los sanitazers: address, leak, undefined. Para graficar las matrices, se procedió a asignar color negro a las casillas con valor 0 en la matriz y generar una paleta de colores para los rangos de los valores de id en la matriz. Se realizó un flat profile del programa para una matriz de tamaño 128 y probabilidad 0.59271 empleando gperf. Finalmente se analizaron los tiempos de ejecución en función del tamaño de la matriz, esto se realizó para dos niveles de optimización: sin optimizar y aplicando la bandera -O3; los resultados fueron normalizados respecto al valor más grande para L=512 sin optimizar.

Análisis y resultados

Inicialmente, el programa se basaba en la recursión directa del algoritmo de búsqueda de clusters, es decir, se llamaba a la función dentro de si misma pero aplicada sobre distintas casillas, adyacentes. Para valores pequeños de L no presentaba problema; sin embargo, para valores más grandes de matriz ej. $L > 250$ A-SAN indica stack-overflow. Posiblemente esto se deba a que el programa entra en recursiones demasiado profundas.

Por ello se completó un algoritmo mas fiel al tipo DFS, en el cual se emplea un buffer y con ello no se hacen varias llamadas anidadas simultáneas a la función. Al compilar con la bandera -Wall aparecen las advertencias: ≪warning: comparison of integer expressions of different signedness: Eigen::Index aka long int and 'std::vector'bool'::size_type aka 'long unsigned int' [-Wsign-compare]≫ y ≪warning: comparison of integer expressions of different signedness: 'int' and 'std::vector ¡bool¿::size_type'



»»Figura 2: Probabilidad de percolación $P(p, L)$ en función de probabilidades de llenado p y diferentes dimensiones L.

»aka 'long unsigned int' [-Wsign-compare]», estos inconvenientes están asociados al tipo de estructuras que se están usando en el programa, su solución requeriría reescribir el programa en términos de un mismo tipo de estructura (en el programa se hace uso de std::vector y Eigen); lo anterior no presenta un peligro grave puesto que se están comparando valores enteros que se espera siempre sean positivos y el límite para el cual la variable tipo signed int admite valores positivos se encuentra fuera del alcance de la aplicación del problema.

No se realizaron pruebas mas allá del tamaño máximo de matriz para el problema o una mayor cantidad de muestras, por lo cual se desconoce los límites a los cuales puede operar el programa. Aunque el programa no parece tener dificultades, en específico un overflow de memoria, con lo cual un aumento, al menos pequeño del tamaño de matriz o de la cantidad de muestras tomadas parece plausible.

El flat-profile (profiling-report.txt) sugiere que el código realizado a pesar de experimentar una gran cantidad de llamadas a las funciones implementadas, no tomaban un tiempo considerablemente grande. Además, de acuerdo con el tiempo empleado para cada función, las funciones toman porcentajes similares de ejecución. Por tal motivo no se consideró necesario hacer modificaciones al código.

Para los tiempos de procesamiento se tiene que la diferencia entre la ejecución del programa compilado sin optimización, contra el compilado con la bandera -O3 es enorme. La dependencia del tiempo de procesamiento respecto al tamaño del sistema se reduce significativamente, por lo cual su uso, sobretodo para matrices muy grandes es fuertemente recomendado. (ver figura 3)

Tabulando la probabilidad de que existiera un cluster percolante en función de la probabilidad de llenado p y el tamaño de la matriz (ver figura 2) se observa que, en general, la probabilidad aumenta abruptamente al llegar a la región de $p = [0{,}5, 0{,}6]$, desde 0 hasta 1. Las desviaciones son notorias dada la naturaleza de la variable analizada: toma valores binarios 0 y 1. La probabilidad crítica en realidad se piensa para sistemas de tamaño infinito [3], por lo cual no se espera obtener un valor exacto, aunque si debe tender a cierto valor al aumentar el tamaño del sistema. Se puede observar que a medida que aumenta el tamaño del sistema el ancho de la curva de aumento de probabilidad se hace menor. Tomando el valor para $L = 512$ en la curva, como el dato más cercano a la probabilidad crítica se tiene que p_c computacional es $p_{c,c} \approx 0{,}59$, cercano al valor de $p_c \approx 0{,}59271$ esperado. Un valor mejor valor podría obtenerse aumentando el tamaño de la matriz, cantidad de tomas en términos de distintos valores de probabilidad alrededor del punto critico, así como la cantidad de muestras en términos de la semilla del generador de valores.

Al graficar las matrices obtenidas para un tamaño y seed fijo, pero probabilidades distintas, cercanas a la probabilidad crítica (ver figura 4), se nota que la estructura general de los clústers no presenta un cambio significativo para $p = 0{,}57$ y $p = 0{,}58$; al llegar a un valor cercano a la probabilidad crítica el tamaño de los clúster aumenta rápidamente – se puede pensar cómo la unión de clústers menores al aparecer nuevas casillas abiertas, que sirven de conexión– hay un desplazamiento de los colores que previamente se encontraban alojados en regiones distintas. Por la forma en la que se generaron las matrices y posteriormente se analizaron los clúster, que los colores sean relativamente homogéneos no implica que haya percolación horizontalmente. Lo anterior se ve de forma más clara al observar la gráfica del tamaño del clúster percolante más grande en función de la probabilidad de llenado. Dado que es natural que, en general, pueda haber clústers más grandes en matrices mas grandes, se realizó la normalización del tamaño del clúster respecto al tamaño de matriz correspondiente. Similar a la probabilidad de formación, el comportamiento se torna más definido, hay cambios más abruptos a medida que aumenta el tamaño de matriz. Nuevamente, matrices más grandes tienen un comportamiento más cercano al hipotético para un

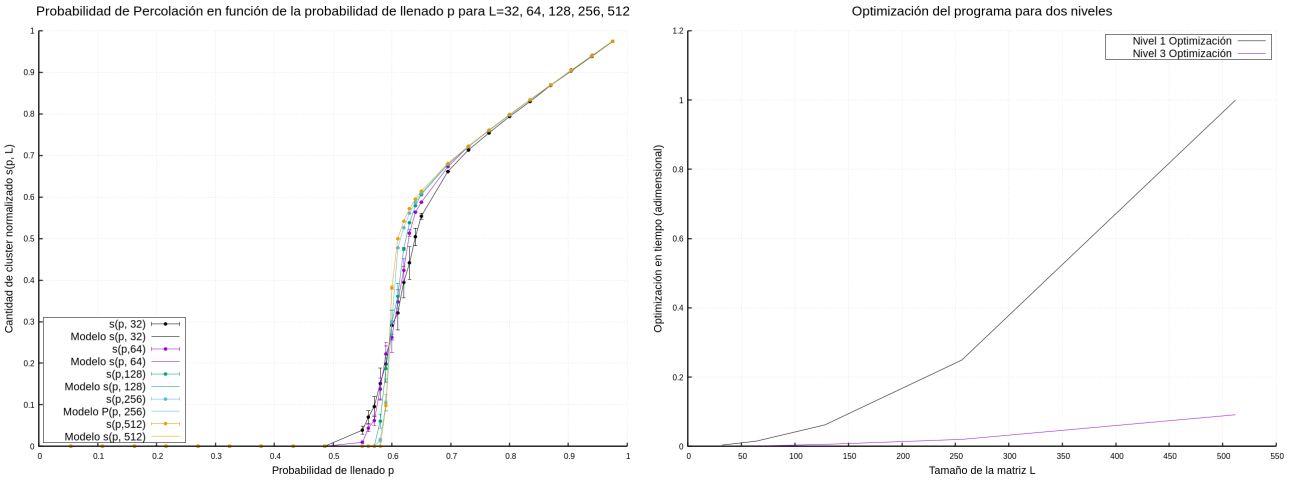


Figura 3: Cantidad de clusters (normalizado) $s(p, L)$ // Tiempo de procesamiento (normalizado) usando dos niveles de optimización.

sistema de tamaño infinito [3]. A grandes rasgos, se puede ver que el tamaño empieza a aumentar repentinamente para valores cercanos a la probabilidad crítica. Después de $p \approx 0{,}7$ las curvas convergen, independientemente del tamaño, y el comportamiento es lineal en dicha región.

Conclusiones

El programa realizado basándose en un algoritmo tipo DFS cumple con los requerimientos del problema satisfactoriamente. Es preferible hacer uso de un buffer para ejecutar el algorítmo que analiza los clusters que usar una recursión directa, puesto que la segunda opción lleva a un stack-overflow a tamaños grandes.

Las matrices de mayor tamaño presentan un comportamiento más definido. Se encuentra que la probabilidad crítica calculada es $p = 0{,}59$. Para probabilidades de llenado menores a esa está la probabilidad de existencia de un clúster percolante es muy baja si no nula. Al alejarse del punto de probabilidad crítica el comportamiento del sistema tiende a ser independiente del tamaño.

Referencias

[1] M. Matsumoto and T. Nishimura, *"Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator"*, ACM Transactions on Modeling and Computer Simulation, vol. 8, no. 1, pp. 3-30, 1998.

[2] MIT, *14. Depth-First Search (DFS), Topological Sort.* 2013 [Online]. Available: `https://www.youtube.com/watch?v=AfSk24UTFS8`. [Accessed: 10 - May- 2020]

[3] G. Grimmett, *Percolation.* Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 1 - 31.

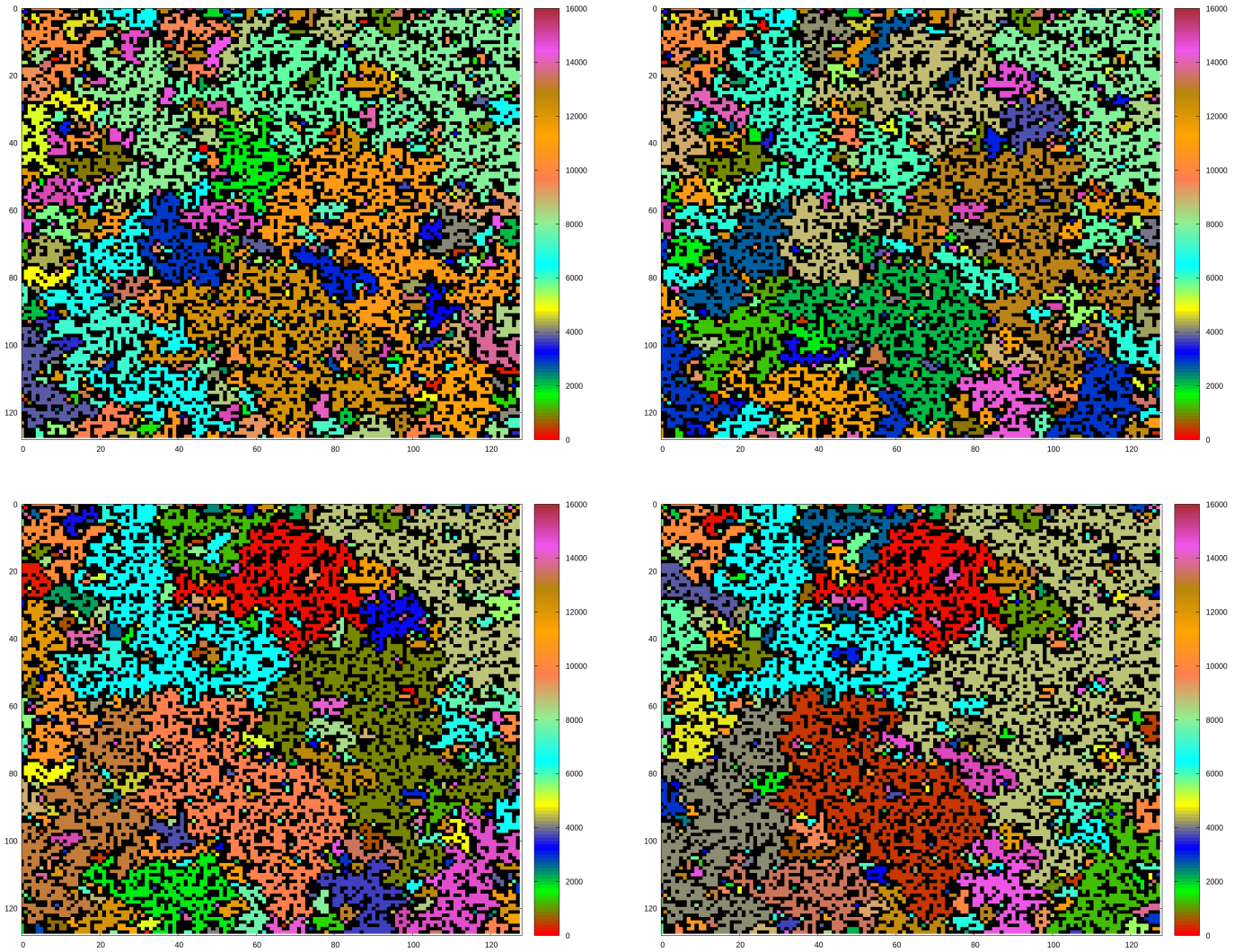


Figure 4: Llenado aleatorio de redes de dimensión $L = 128$ para p=0.57, p=0.58, p=0.59, p=0.60.

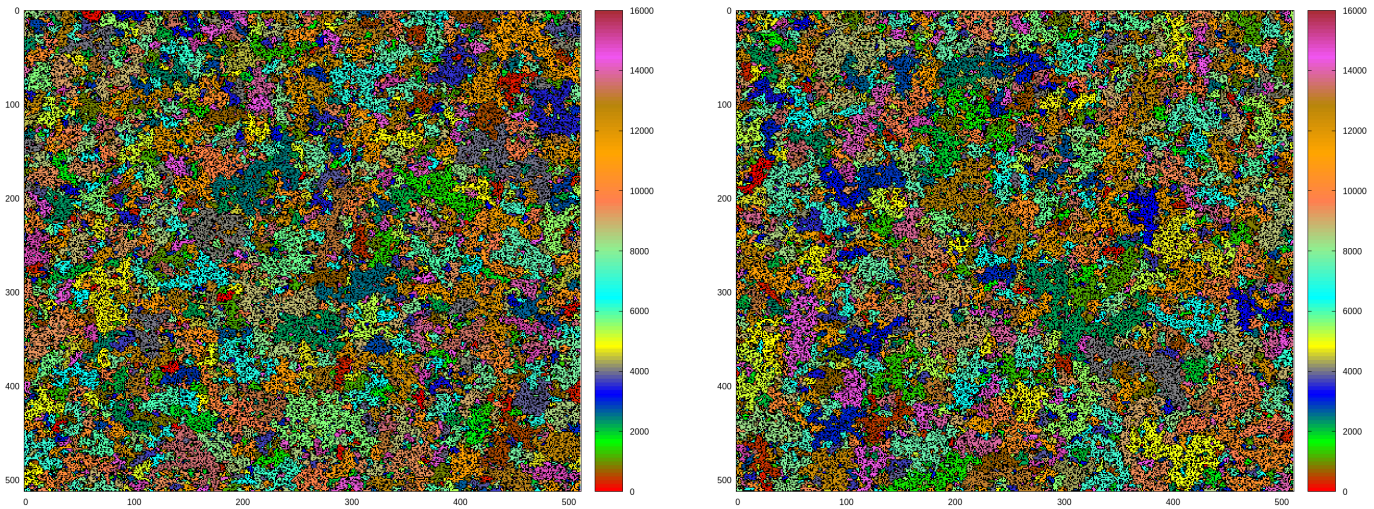


Figure 5: Llenado aleatorio de redes de dimensión $L = 512$ para dos seeds diferentes (9, 11) y una misma probabilidad.