Tarea 2
Simulación de Sistemas

Beatriz Alejandra García Ramos

A 21 de Agosto de 2017

1 Autómata celular

Utilizando la finalidad del juego de la vida representamos si una celda vive o muere.
Las celdas vivas son las que contienen un 1 y las muertas un 0. Cuando una celda viva tiene alrededor más o menos de 3 vecinos vivos entonces muere, si tiene exactamente 3 vecinos vivos sigue viviendo. Dado que al generar las celdas vivientes se utiliza una distribución uniformemente al azar se tiene una probabilidad de 0.5 de aparecer en nuestra matriz.

2 Tareas

- Determinar el número de iteraciones que dura la simulación en función de la probabilidad inicial de celda viva.
- Modificar la simulación para que modele un tipo de crecimiento donde se crean núcleos que inician en celdas aleatorias y éstos se van expandiendo.
- Modificar lo anterior para que nuevos núcleos aparezcan en momentos distintos, no sólo al inicio, en celdas que no se hayan ocupado.

3 Solución

3.1 Tarea base

Para determinar el número de iteraciones que dura la simulación dependiendo de la probabilidad inicial de celda viva primero investigué sobre cómo podría lograr que el programa iniciara con distintas probabilidades, lo que encontré fue que podía añadir un **for** donde se tomara un vector (llamado *proba*) creado desde 0.1 hasta 0.9 en donde la diferencia entre cada elemento fuera de 0.1, así que existen 9 probabilidades distintas donde el código genera datos y al final los compara en una *boxplot* (nótese que pueden ser más probabilidades dependiendo de los valores que le demos a *proba*).

No es necesario saber lo que sucede con la matriz original o el cambio que hace cada vez que ocuure una iteración (además sería complicado tener esas imágenes por cada probabilidad dada) así que lo que se relacionaba con la generación de imágenes fue eliminado, sin embargo se creó un vector llamado *vector*, donde, después de haber creado un **for** que me repita todo el código las veces que yo quiera, se guarde el número de iteraciones que se llegó a hacer cada vez que yo repetí el código; lo cual nos permite saber para cada probabilidad en promedio cuántas iteraciones se realizan (tomando muestras de 15 corridas para cada probabilidad en este caso).

Dado que el vector *datos* recopila los valores de *vector* ahora sí creamos una imagen en donde obtengamos nuestra *boxplot* que nos arroja como eje de abscisas nuestras probabilidades y como eje de ordenadas nuestras iteraciones para cada probabilidad. En ella podemos observar que la media de las

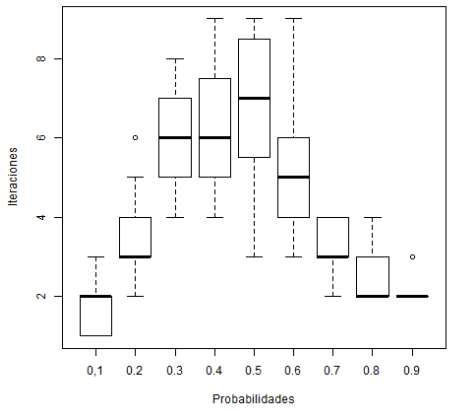

Figure 1: Iteraciones en función de la probabilidad de inicio.

probabilidades se distribuye de manera normal (tiene forma de "campana"), lo cual se esperaba ya que utilizando la lógica de nuestro código mientras más celdas estén muertas al inicio habrá muy pocas o ninguna viva en nuestra primer iteracion, de la misma manera para el caso en el que la mayoría de las celdas estén vivas. La media se encontró en este caso en probabilidad de 0.5, como se observa en la Figura 1, lo cual esperaba porque en el caso en el que la probabilidad toma ese valor casi la mitad de las celdas están vivas pero distribuídas aleatoriamente así que habrá algunas que vivirán durante más iteraciones.

3.2 Reto 1

Añadí un vector *num.nucleo* en donde le digo la cantidad de núcleos con los que deseo iniciar mi matriz, la cual está generada por ceros. Elijo una cantidad de *num.nucleo* números de manera aleatoria y redondeados para que me permita elegir en qué celda un cero se va a modificar por un núcleo. Una vez que tengo esos números en mi vector *a* con un **for** desde 1 hasta *num.nucleo* sustituyo los ceros.

Para poder trabajar con este reto tuve que hacer un cambio en la función paso, ya que ésta me ayuda a observar las vecindades de mis celdas. Dentro de la función utilicé una condición **if** que me permite ver si mi celda es un cero o si no; cuando no es un cero la función paso "regresa" el mismo valor de la celda (es decir, el núcleo encontrado), pero si es un cero lo que hace es "observar" la vecindad y si alguna celda de la vecindad es diferente de cero entonces en el vector *crear*, que fue añadido antes de la condición, se agregan los valores de las celdas distintas de cero, luego se toma el valor del primer elemento (se toma el primero porque aunque puede haber muchos nucleos alrededor, al inicio es poco probable y se podría tener solamente 1 valor en el vector *crear*) y eso es lo que "regresa" paso.

Ahora, como el resto del código es casi el mismo que el original (salvo que cambié algunos nombres) lo que va a suceder es que *parSapply* va a tomar mi función paso desde $pos = 1$ hasta $pos = num$ (que en este caso son 900) y las observaciones de las vecindades ocurren al mismo tiempo cada vez que se hace un paso. Así si hay celdas "vacías" al rededor de un núcleo (en su vecindad) éstas toman el valor del núcleo (nótese que las celdas no se sobreescriben porque la condición nos dice que se van a llenar las celdas que son iguales a cero, mencionado anteriormente). Para poder visualizar lo que hace el código pedí que si se mandaran guardar las imágenes de cada iteración para crear un gif.

Anexo en GitHub se encontrará el gif que nos permitirá observar el crecimiento de un código en el que tenemos dimensión 30 y se crean 5 núcleos iniciales.

Ambos códigos también se encuentran anexados en GitHub.