Práctica 6

Jose David Martinez Ruiz

19 de septiembre de 2017

1. Introducción

Esta práctica consiste en identificar las áreas en donde se puede implementar la paralelización de un programa. El programa en cuestión se trata de un sistema multiagente, en donde se estará propagando una enfermedad con una probabilidad p_i. Los estados de cada agente son "S" que quiere decir que se encuentra susceptible a la contraer la enfermedad, "I" si se encuentra infectado por la enfermedad. Cada agente infectado tiene una probabilidad p_r de recuperarse, al recuperarse no puede contagiar la enfermedad ni contagiarse de nuevo. Cuando se recuperan el estado del agente pasa a ser "R". Cada agente se moverá con una velocidad constante en un toro formado a partir de un rectángulo. La forma de propagación de la enfermedad es que cuando la distancia que hay entre un agente infectado y un agente susceptible es lo suficientemente pequeña.

Se espera que al paralelizar el código el tiempo de ejecución resulte más pequeño, para corroborarlo se tomó el tiempo de ejecución ejecutando la paralelización y se comparó con el tiempo de ejecución del código original.

Para el reto uno, una vez habiendo implementado la paralización se procedió a variar el valor de la probabilidad de infección y se realizaron varias réplicas de la simulación y se comparó el resultado de ellas.

Para el resto dos, al código ya paralelizado se procedió a agregar una probabilidad de que los agentes se encuentren vacunados al iniciar la simulación, se varió el valor de la probabilidad vacunación y se tomaron varias réplicas para cada caso.

2. Desarrollo

Para esta práctica nos interesa identificar las áreas del código donde se puede implementar la paralelización. Para esto se examinó cuidadosamente el cogido. Se espera que al paralelizar el código el tiempo de ejecución resulte más pequeño. Se comparó los tiempos de ejecución implementando la paralelización en cada una de las áreas que se encontraron y se compararon con el tiempo del código original.

Primero se analizó el código, buscando áreas donde se pudiera implementar la paralelización, se encontraron dos áreas en el código. La primera era cuando los agentes se iban contagiando al acercarse a un agente infectado y la segunda es al actualizar los estados y cambiar las distancias para cada agente. Observando lo que hacía el código originalmente en esas dos áreas en particular, se procedió a hacer una función que pudiera realizar las tareas para esas áreas del código. Una función para cada área.

```
actualizar <- function(){
  a <- agentes[i, ]
  if (contagios[i] & a$estado == "S") {
    a$estado <- "I"
  } else if (a$estado == "I") {
    if (runif(1) < pr) {
      a$estado <- "R"
```

```
    }
  }
  a$x <- a$x + a$dx
  a$y <- a$y + a$dy
  if (a$x > l) {
    a$x <- a$x - l
  }
  if (a$y > l) {
    a$y <- a$y - l
  }
  if (a$x < 0) {
    a$x <- a$x + l
  }
  if (a$y < 0) {
    a$y <- a$y + l
  }
  return(as.vector(a))
}

contagiar <- function(){
  if (!contagios[j]) { # aun sin contagio
    a2 <- agentes[j, ]
    if (a2$estado == "S") { # hacia los susceptibles
      dx <- a1$x - a2$x
      dy <- a1$y - a2$y
      d <- sqrt(dx^2 + dy^2)
      if (d < r) { # umbral
        p <- (r - d) / r
        if (runif(1) < p) {
          return(TRUE)
        }
        else{
          return(FALSE)
        }
      }
      return(FALSE)
    }
    else if(a2$estado == "I" || a2$estado == "R") {
      return(TRUE)
    }
  }
  else{
    return(TRUE)
  }
}
```

Para paralelizar dichas funciones, primero se utilizó la librería doParallel y se revisó cuidadosamente que las funciones ya mencionadas funcionaran correctamente. Posteriormente se intentó paralelizar utilizando la librería Parallel, al ejecutar el programa resultó en que los agentes infectados no propagaban la enfermedad. Se realizaron diversas modificaciones intentando hacer que el programa funcionara, pero debido a mi falta de precisión de conocimientos sobre la aplicación e implementación de la librería Parallel, no me fue posible implementar esa paralelización, por lo que se trabajará el resto de la practica utilizando para paralelizar la librería doParallel.

Al comparar el tiempo del código original y del código paralelizado, se obtuvo como resultado que al código paralelizado le toma más tiempo que el código original, 97.83 segundos más lento que el original. Debido a que esto fue un resultado extraño se procedió a verificar qué modificación que se realizó provoca el aumento de tiempo. Para esto, se comparó el tiempo que le tomo al código paralelizado sólo en una de las dos áreas.

Se hizo un código utilizando la paralelización sólo en la función contagiar y otro código utilizando la paralelización en la función actualizar. Para estos dos casos se tomó el tiempo de ejecución y se comparó con el tiempo del código original. Esto dio como resultado que ambas implementaciones se tardan más que el código original, aunque la paralelización de la función actualizar es más rápida que la paralelización de la función contagiar. Esto se debe a que por cómo está estructurado la parte del código que dice quién se va a contagiar, se encuentra dentro de un for la función contagiar paralelizada, por lo que realizar esas acciones toma más tiempo y esfuerzo computacional que solamente implementar una sola función en paralelo.

(En el repositorio de Github se adjuntara el código para cada paralelizacion, siendo p6.R el código con las 2 paralelizaciones, p6a.R el código paralelizando la función actualizar y p6b.R el código paralelizando la función contagiar.)

3. Reto 1

Para el primer reto que consistía en agregar una variable nueva que representara la probabilidad de que los agentes se vacunen contra la enfermedad al iniciar la simulación, lo que se hizo fue probar diferentes probabilidades de vacunación para así poder ver cómo se comporta la simulación con agentes vacunados. Las probabilidades fueron variando desde el 5 hasta una probabilidad del noventa y cinco por ciento. Los porcentajes máximos para cada probabilidad de observa en la siguiente tabla

Probabilidades	Máximos
0.05	72
0.10	66
0.15	62
0.20	58
0.25	52
0.30	58
0.35	48
0.40	52
0.45	44
0.50	52
0.55	42
0.60	40
0.65	42
0.70	36
0.75	34
0.80	34
0.85	38
0.90	28
0.95	34

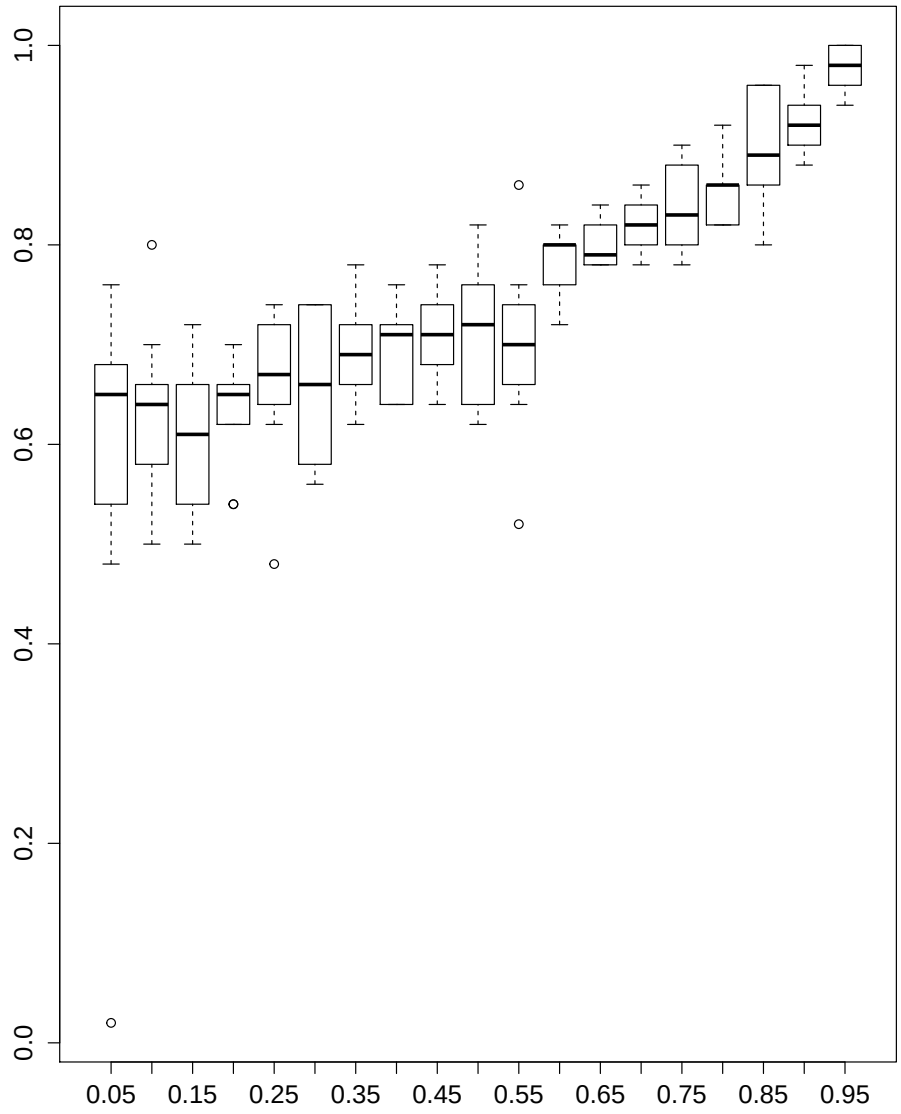
Cuadro 1: Porcentaje máximo de infectados en cada probabilidad.

Como se puede observar los porcentajes maximos de infectados tiende a disminuir conforme la probabilidad de vacunación es más grande. Sigue habiendo infectados pero la vacuna no les permite abarcar gran porcentaje de la población

4. Reto 1

Para el segundo reto consistía en variar la probabilidad de que los agentes se infectan de la enfermedad cuando inicia la simulación, para poder hacer eso se varió desde una probabilidad del

cinco por ciento hasta el noventa y cinco por ciento. Se hicieron diez repeticiones para cada probabilidad, los resultados se muestran en la siguiente caja de bigotes



Como se puede observar el porcentaje de infectados cada vez se va disparando conforme la probabilidad inicial de infectados aumenta

5. Conclusiones

De la tarea base se concluye que debido a que la paralelización fue implementada dentro de un ciclo, fue por eso que la implementación tomo mas tiempo que el código original. Es probable que en al usar el paralelismo se tarde más que el código original, debido a que como no son iteraciones muy grandes, resulte más fácil para la computadora ejecutarlo secuencialmente que utilizando núcleos para ejecutarlo

del primer reto se concluye que la vacunación hace que los el porcentaje de infectados sea ca-

da vez menor, ya que los agentes vacunados no propagan la enfermedad ni tampoco la contraen, es por eso que el porcentaje máximo de infectados tiende a bajar, inclusive hubo porcentajes cercanos o iguales al cero.

Del reto dos se concluye que debido a que la probabilidad de infectados va en aumento, también el porcentaje máximo de infectados, esto es debido que, inicialmente, puede haber una gran cantidad de infectados por lo que la enfermedad abarca con más facilidad el total de la población.