Tarea 6: sistema multiagente

Simulación de Sistemas

18 de septiembre de 2017

1. Introducción

Un sistema multiagente de manera general es un conjunto de entidades inteligentes que tienen comunicación e interactúan entre sí.

Se implementa un sistema multiagente para simular una epidemia y los estados que pueden tomar los agentes son: *susceptibles* (S), *infectados* (I) o *recuperados* (R). Los parámetros que se consideran son cantidad de agentes n, probabilidad inicial de infección p_i, probabilidad de recuperarse p_r y umbral r. Esta tarea consiste en paralelizar el código proporcionado, así mismo hacer los cambios necesarios para realizar la simulación.

2. Tarea

La siguiente tarea se realizó en una máquina con las siguientes especificaciones: procesador Intel(R)Core(TM) i5-6200U CPU 2.30 GHz 2.40 GHz con 8GB en memoria RAM y sistema operativo Windows 10 Home. Se emplearon tres de los cuatro núcleos.

Para paralelizar la simulación de la práctica 6 hay que hacer varias modificaciones como independizar ciertas partes del código en diferentes funciones. Las cuatro funciones creadas son: `contagiado, recuperado, movimientox` y `movimientoy`.

La función `contagiado` considera los agentes con estado S y corrobora si el resto de los agentes con estado I tienen una distancia hacia él menor al umbral, de ser así con una probabilidad p_i cambia su estado a I. La función `recuperado` determina si un agente que se encuentra en estado I cambia a estado R en el siguiente movimiento con una probabilidad p_r. Las funciones `movimientox` y `movimientoy` actualizan la posición del agente en cada paso.

Se muestra a continuación la función `contagiado`.

```
contagiado <- function(i){
  a1 <- agentes[i, ]
  if(a1$estado == "S"){#es susceptible
       for(k in quiencontagio){
         a2 <- agentes[k, ]
         dx <- a1$x - a2$x
         dy <- a1$y - a2$y
         d <- sqrt(dx^2 + dy^2)
         if(d < r){#tiene distancia menor que el umbral
            p <- (r-d)/r
            if(runif(1) < p){#cumple probabilidad
              return("I")
        }
      }
    }
    return("S")
  }
  if(a1$estado == "I"){#ya estaba infectado
    return("I")
  }
  if(a1$estado == "R"){# esta recuperado
    return("R")
  }
}
```

Para esta tarea se utilizó la librería `parallel` de R y las funciones a paralelizar son las anteriormente mencionadas. Se presentará cómo se paralelizó la función `contagiado` y de manera similar se realizó para el resto de las funciones.

```
  cluster <- makeCluster(detectCores()-1)
  clusterExport(cluster, "contagiado")
  clusterExport(cluster, "n")
  clusterExport(cluster, "agentes")
  clusterExport(cluster, "quiencontagio")
  clusterExport(cluster, "r")
  vecnvoscontagios <- parSapply(cluster, 1:n, contagiado)
  stopCluster(cluster)
```

Se calculó el tiempo para veinte ejecuciones del código en forma secuencial, el valor promedio de tiempo es 6.454571. Se realizó lo mismo para el código donde se implemento paralelismo dando como promedio de tiempo 3,085134. La diferencia en los tiempos es notoria, con lo que se concluye que implementar el código de manera paralela es bastante útil cuando se esta interasado en el tiempo, ya que en este caso se el código de manera secuencial tomó el doble de tiempo que el de manera paralela.

3. Reto 1

En el primer reto se desea tener agentes vacunados desde el comienzo de la simulación con probabilidad p_v. Para esto, en la estructura *agentes* se determina con probabilidad p_v si un agente es vacunado, si es así su estado es R o de otra manera su estado será S. Una vez que todos los agentes tienen estado ya sea R o S, se consideran solo aquellos con estado S y a partir de ellos se decide con probabilidad p_i si su estado cambia a I.

La parte modificada se muestra a continuación.

```
for (i in 1:n) {
  e <- "S"
  if (runif(1) < pv) {
    e <- "R"
  }
  agentes <- rbind(agentes, data.frame(x = runif(1, 0, l),
                                       y = runif(1, 0, l),
                                      dx = runif(1, -v, v),
                                      dy = runif(1, -v, v),
                                      estado = e))
  levels(agentes$estado) <- c("S", "I", "R")
}

for(i in 1:n){
  vac <- agentes[i, ]
  if(vac$estado == "S"){
    if(runif(1)< pi){
      vac$estado <- "I"
      agentes[i, ] <- vac
    }
  }
}
```

Se varió la probabilidad de vacunados p_v de 0.1 a 0.5 con incrementos de 0.1, cada uno con 30 repeticiones, dejando fijo los parámetros $n = 50$, $p_i = 0.05$ y $p_r = 0.02$. En la figura 1 se muestran los resultados.

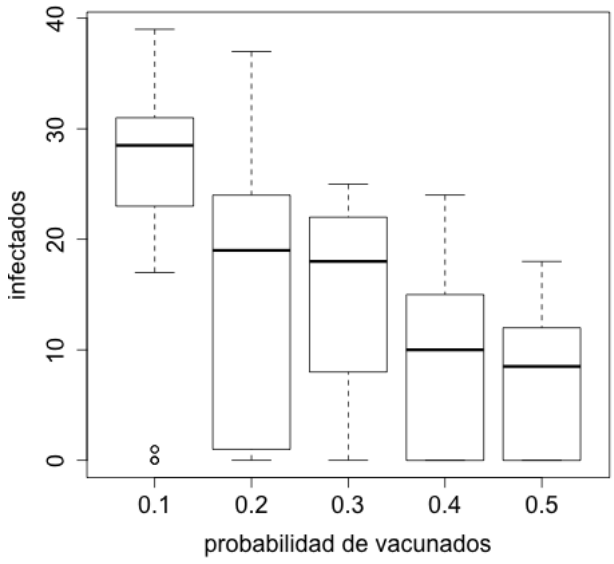

Figura 1: Porcentaje máximo de infectados durante la simulación para probabilidad de vacunados que va de 0.1 a 0.5.

Notemos en la figura 1 que conforme aumenta la probabilidad de agentes vacunados disminuye la cantidad de agentes infectados. Las medias para $p_v = 0.2$ y $p_v = 0.3$ toman un valor muy cercano a veite unidades, mientras que las medias de $p_v = 0.4$ y $p_v = 0.5$ apenas sobrepasan el valor de 10 unidades.

4. Reto 2

Para el segundo reto se pide examinar el efecto que hay entre la probabilidad de infectados p_i y el porcentaje de infectados. Para lo cual se varió el valor de p_i de 0.05 a 0.5 con incrementos de 0.05, cada uno con 15 repeticiones. Donde se tiene fijo el número de agentes $n = 50$ y la probabilidad de recuperación $p_r = 0.02$. En la figura 2 se muestran los resultados obtenidos.

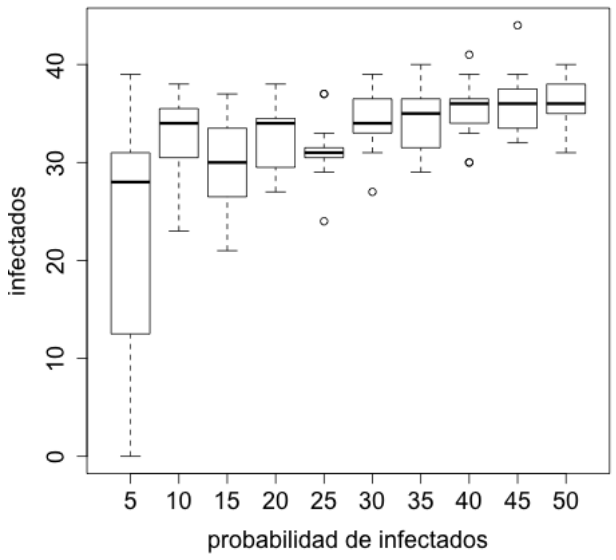

Figura 2: Porcentaje máximo de infectados durante la simulación para probabilidades que van desde 0.05 hasta 0.5.

Como se puede ver la cantidad de infectados es menor cuando $p_i = 0{,}05$ cuya media se mantiene abajo de las treinta unidades; sin embargo, la cantidad de infectados aumenta conforme aumenta p_i. Notemos en la figura 2 que cuando $p_i = 0{,}25$ los infectados se mantienen muy cerca de treinta unidades pero para probabilidades entre 0,3 y 0,5 la cantidad de afectados aumenta pero la media en este intervalo no pasa las cuarenta unidades.