Reporte Práctica 4

Diagramas de Voronoi

Juan Pablo Chávez Granados

A 5 de Septiembre de 2017

Diagramas de Voronoi

Los diagramas de Voronoi surgen de la unión de varios polígonos conocidos como polígono de Voronoi, éste es el conjunto de todos los puntos más cercanos a un punto dado.1 En matemática los diagramas de Voronoi se definen como la división de un plano en regiones basadas en la distancia varios puntos en un orden específico sobre el plano.2

El uso de los Diagramas de Voronoi se ha vuelto tan extenso que tiene aplicaciones en todos los campos de la ciencia, en las ciencias naturales los diagramas de Voronoi se utilizan en la biología, hidrología, ecología, química computacional, astrofísica, dinámica de fluidos computacional y en la física computacional. En la medicina se utiliza en el diagnóstico médico y en epidemiología. En la ingeniería se utiliza en la física de polímeros, ciencia de materiales, aviación, arquitectura y minería, éstos son algunos de los campos en los que se aplican los diagramas de voronoi.

Práctica

En la práctica de diagramas de Voronoi se examinó el efecto que el número de semillas y el tamaño de la zona tienen en los largos de las grietas que se forman.

```
- n <-  40
- zona <- matrix(rep(0, n * n), nrow = n, ncol = n)
- k <- 12
- x <- rep(0, k) # ocupamos almacenar las coordenadas x de las semillas
- y <- rep(0, k) # igual como las coordenadas y de las semillas
-
- for (semilla in 1:k) {
-     while (TRUE) { # hasta que hallamos una posicion vacia para la semilla
-         fila <- sample(1:n, 1)
-         columna <- sample(1:n, 1)
-         if (zona[fila, columna] == 0) {
-             zona[fila, columna] = semilla
-             x[semilla] <- columna
-             y[semilla] <- fila
-             break
-         }
-     }
- }
-
```

```
-   celda <-  function(pos) {
-      fila <- floor((pos - 1) / n) + 1
-      columna <- ((pos - 1) %% n) + 1
-      if (zona[fila, columna] > 0) { # es una semilla
-         return(zona[fila, columna])
-      } else {
-         cercano <- NULL # sin valor por el momento
-         menor <- n * sqrt(2) # mayor posible para comenzar la busqueda
-         for (semilla in 1:k) {
-            dx <- columna - x[semilla]
-            dy <- fila - y[semilla]
-            dist <- sqrt(dx^2 + dy^2)
-            if (dist < menor) {
-               cercano <- semilla
-               menor <- dist
-            }
-         }
-         return(cercano)
-      }
-   }
-
-   suppressMessages(library(doParallel))
-   registerDoParallel(makeCluster(detectCores() - 1))
-   celdas <- foreach(p = 1:(n * n), .combine=c) %dopar% celda(p)
-   stopImplicitCluster()
-   voronoi <- matrix(celdas, nrow = n, ncol = n, byrow=TRUE)
-   rotate <- function(x) t(apply(x, 2, rev))
-   png("p4s.png")
-   par(mar = c(0,0,0,0))
-   image(rotate(zona), col=rainbow(k+1), xaxt='n', yaxt='n')
-   graphics.off()
-   png("p4c.png")
-   par(mar = c(0,0,0,0))
-   image(rotate(voronoi), col=rainbow(k+1), xaxt='n', yaxt='n')
-   graphics.off()
-
-   limite <- n # grietas de que largo minimo queremos graficar
-
-   inicio <- function() {
-      direccion <- sample(1:4, 1)
-      xg <- NULL
-      yg <- NULL
-      if (direccion == 1) { # vertical
```

```
-           xg <- 1
-           yg <- sample(1:n, 1)
-       } else if (direccion == 2) { # horiz izr -> der
-           xg <- sample(1:n, 1)
-           yg <- 1
-       } else if (direccion == 3) { # horiz der -> izq
-           xg <- n
-           yg <- sample(1:n, 1)
-       } else { # vertical al reves
-           xg <- sample(1:n, 1)
-           yg <- n
-       }
-       return(c(xg, yg))
-   }
-
-   vp <- data.frame(numeric(), numeric()) # posiciones de posibles vecinos
-   for (dx in -1:1) {
-       for (dy in -1:1) {
-           if (dx != 0 | dy != 0) { # descartar la posicion misma
-               vp <- rbind(vp, c(dx, dy))
-           }
-       }
-   }
-   names(vp) <- c("dx", "dy")
-   vc <- dim(vp)[1]
-
-   propaga <- function(replica) {
-       # probabilidad de propagacion interna
-       prob <- 1
-       dificil <- 0.99
-       grieta <- voronoi # marcamos la grieta en una copia
-       i <- inicio() # posicion inicial al azar
-       xg <- i[1]
-       yg <- i[2]
-       largo <- 0
-       while (TRUE) { # hasta que la propagacion termine
-           grieta[yg, xg] <- 0 # usamos el cero para marcar la grieta
-           largo <-  largo + 1
-           frontera <- numeric()
-           interior <- numeric()
-           for (v in 1:vc) {
-               vecino <- vp[v,]
-               xs <- xg + vecino$dx # columna del vecino potencial
```

```
-             ys <- yg + vecino$dy # fila del vecino potencial
-             if (xs > 0 & xs <= n & ys > 0 & ys <= n) { # no sale de la zona
-                if (grieta[ys, xs] > 0) { # aun no hay grieta ahi
-                   if (voronoi[yg, xg] == voronoi[ys, xs]) {
-                      interior <- c(interior, v)
-                   } else { # frontera
-                      frontera <- c(frontera, v)
-                   }
-                }
-             }
-          }
-          elegido <- 0
-          if (length(frontera) > 0) { # siempre tomamos frontera cuando haya
-             if (length(frontera) > 1) {
-                elegido <- sample(frontera, 1)
-             } else {
-                elegido <- frontera # sample sirve con un solo elemento
-             }
-             prob <- 1 # estamos nuevamente en la frontera
-          } else if (length(interior) > 0) { # no hubo frontera para propagar
-             if (runif(1) < prob) { # intentamos en el interior
-                if (length(interior) > 1) {
-                   elegido <- sample(interior, 1)
-                } else {
-                   elegido <- interior
-                }
-                prob <- dificil * prob # mas dificil a la siguiente
-             }
-          }
-          if (elegido > 0) { # si se va a propagar
-             vecino <- vp[elegido,]
-             xg <- xg + vecino$dx
-             yg <- yg + vecino$dy
-          } else {
-             break # ya no se propaga
-          }
-       }
-       if (largo >= limite) {
-          png(paste("p4g_", replica, ".png", sep=""))
-          par(mar = c(0,0,0,0))
-          image(rotate(grieta), col=rainbow(k+1), xaxt='n', yaxt='n')
-          graphics.off()
-       }
```

```
-       return(largo)
-   }
-   #for (r in 1:10) { # para pruebas sin paralelismo
-   #    propaga(r)
-   #}
-   suppressMessages(library(doParallel))
-   registerDoParallel(makeCluster(detectCores() - 1))
-   largos <- foreach(r = 1:200, .combine=c) %dopar% propaga(r)
-   stopImplicitCluster()
-   summary(largos)Código inicial
```

Este código genera semillas o puntos aleatorios los cuales se expanden con una función celda integrada en el código, ésta se utilizó para expandir las dimensiones de la semilla, también se tienen una función propaga la cual genera una grita en una posición al azar y hace que vaya expandiéndose hasta perder la energía para seguir propagándose.

Este código se modificó para poder llevar a cabo la examinación del efecto de las dimensiones de la celda y el número de semillas en el experimento.

- Código Pablo

```
k <- 10

rg = 20

suppressMessages(library(doParallel))

registerDoParallel(makeCluster(2))

largos <- data.frame()

for(n in c(40, 50, 60)) {


  zona <- matrix(rep(0, n * n), nrow = n, ncol = n)

  x <- rep(0, k) # ocupamos almacenar las coordenadas x de las semillas

  y <- rep(0, k) # igual como las coordenadas y de las semillas


  for (semilla in 1:k) {

    while (TRUE) { # hasta que hallamos una posicion vacia para la semilla

      fila <- sample(1:n, 1)

      columna <- sample(1:n, 1)
```

```
    if (zona[fila, columna] == 0) {
      zona[fila, columna] = semilla
      x[semilla] <- columna
      y[semilla] <- fila
      break
    }
  }
}

celda <-  function(pos) {
  fila <- floor((pos - 1) / n) + 1
  columna <- ((pos - 1) %% n) + 1
  if (zona[fila, columna] > 0) { # es una semilla
    return(zona[fila, columna])
  } else {
    cercano <- NULL # sin valor por el momento
    menor <- n * sqrt(2) # mayor posible para comenzar la busqueda
    for (semilla in 1:k) {
      dx <- columna - x[semilla]
      dy <- fila - y[semilla]
      dist <- sqrt(dx^2 + dy^2)
      if (dist < menor) {
        cercano <- semilla
        menor <- dist
      }
    }
    return(cercano)
  }
```

```
}


celdas <- foreach(p = 1:(n * n), .combine=c) %dopar% celda(p)
voronoi <- matrix(celdas, nrow = n, ncol = n, byrow=TRUE)
inicio <- function() {
  direccion <- sample(1:4, 1)
  xg <- NULL
  yg <- NULL
  if (direccion == 1) { # vertical
    xg <- 1
    yg <- sample(1:n, 1)
  } else if (direccion == 2) { # horiz izr -> der
    xg <- sample(1:n, 1)
    yg <- 1
  } else if (direccion == 3) { # horiz der -> izq
    xg <- n
    yg <- sample(1:n, 1)
  } else { # vertical al reves
    xg <- sample(1:n, 1)
    yg <- n
  }
  return(c(xg, yg))
}

vp <- data.frame(numeric(), numeric()) # posiciones de posibles vecinos
for (dx in -1:1) {
  for (dy in -1:1) {
```

```
    if (dx != 0 | dy != 0) { # descartar la posicion misma
      vp <- rbind(vp, c(dx, dy))
    }
  }
}
names(vp) <- c("dx", "dy")
vc <- dim(vp)[1]

propaga <- function(replica) {
  # probabilidad de propagacion interna
  prob <- 1
  dificil <- 0.99
  grieta <- voronoi # marcamos la grieta en una copia
  i <- inicio() # posicion inicial al azar
  xg <- i[1]
  yg <- i[2]
  largo <- 0
  while (TRUE) { # hasta que la propagacion termine
    grieta[yg, xg] <- 0 # usamos el cero para marcar la grieta
    largo <-  largo + 1
    frontera <- numeric()
    interior <- numeric()
    for (v in 1:vc) {
      vecino <- vp[v,]
      xs <- xg + vecino$dx # columna del vecino potencial
      ys <- yg + vecino$dy # fila del vecino potencial
      if (xs > 0 & xs <= n & ys > 0 & ys <= n) { # no sale de la zona
        if (grieta[ys, xs] > 0) { # aun no hay grieta ahi
```

```
      if (voronoi[yg, xg] == voronoi[ys, xs]) {
        interior <- c(interior, v)
      } else { # frontera
        frontera <- c(frontera, v)
      }
    }
  }
}
elegido <- 0
if (length(frontera) > 0) { # siempre tomamos frontera cuando haya
  if (length(frontera) > 1) {
    elegido <- sample(frontera, 1)
  } else {
    elegido <- frontera # sample sirve con un solo elemento
  }
  prob <- 1 # estamos nuevamente en la frontera
} else if (length(interior) > 0) { # no hubo frontera para propagar
  if (runif(1) < prob) { # intentamos en el interior
    if (length(interior) > 1) {
      elegido <- sample(interior, 1)
    } else {
      elegido <- interior
    }
    prob <- dificil * prob # mas dificil a la siguiente
  }
}
if (elegido > 0) { # si se va a propagar
  vecino <- vp[elegido,]
```

```
      xg <- xg + vecino$dx
      yg <- yg + vecino$dy
    } else {
      break # ya no se propaga
    }
  }
  return(largo)
 }
 largos = rbind(largos, foreach(r = 1:rg, .combine=c) %dopar% propaga(r))
}
stopImplicitCluster()
n40 = as.numeric(largos[1, ])
n50 = as.numeric(largos[2, ])
n60 = as.numeric(largos[3, ])
labels = c("n = 40", "n = 50", "n = 60")
boxplot(n40, n50, n60, col=c(2,3,4), names = labels, ylab="Largo de grieta", xlab="Celdas", ylim = c(1, 200))
```

En este código se conservó activo el paquete “doParallel”, el cual permite hacer múltiples cálculos haciendo uso de los núcleos en la computadora, también permite dar múltiples ordenamientos para el uso de esos núcleos. En el código se retiraron las líneas de comando que imprimían las matrices con la propagación de la grieta, también se retiraron las líneas que generaban varios clusters y sólo se dejó una línea de generación del cluster y un stopCluster para detenerlo, al final del código se generó un boxplot en el cual se presentan tres tamaños de celda para un mismo número de semillas.

Resultados

Como resultado de la examinación se obtuvieron tres diagramas de caja pivote cada uno para diferente número de semillas en las celdas. En la figura 1 tenemos el diagrama caja pivote para 10 semillas en la celda con dimensiones de 40x40, 50x50 y 60x60 respectivamente, en este se puede observar como el largo de las grietas es mayor para la celda con dimensiones de 40x40, mientras que para las celdas con dimensiones mayores el largo de las grietas es menor.

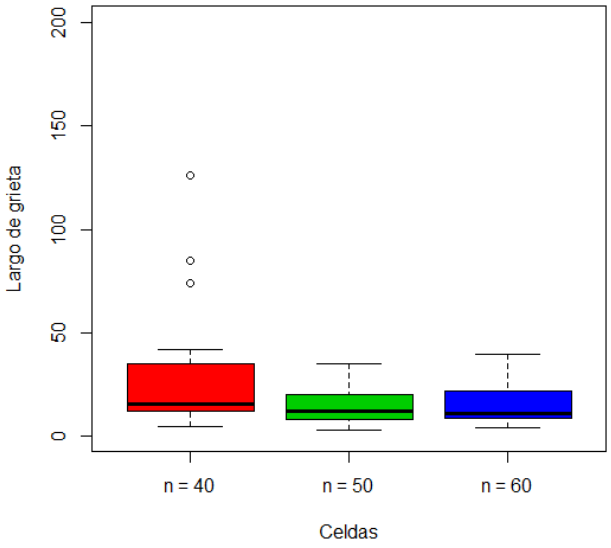

Fig. 1. Diagrama caja pivote para una k=10.

En la figura 2 que es el diagrama caja pivote para un número de semillas de 20 se puede apreciar que al aumentar el número de semillas en la celda aumenta la longitud de las grietas para la celda con dimensiones de 40x40, mientras que en las celdas con dimensiones mayores no se considera que el efecto se apreciable por el número de semillas que se manejan.

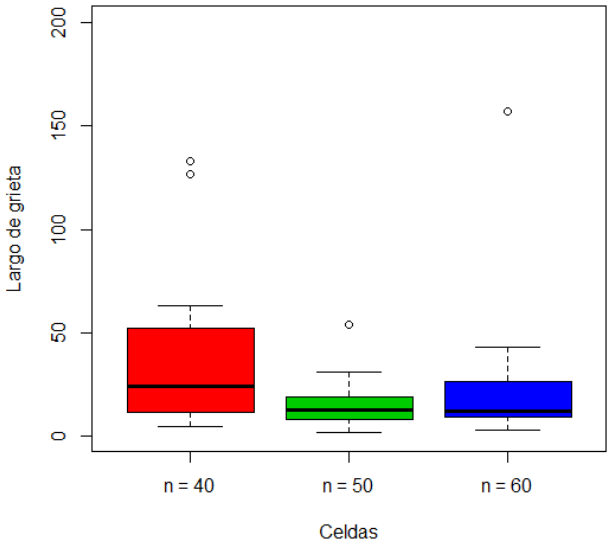

Fig. 2. Diagrama caja pivote para una k=20.

En la figura 3 se aprecia el diagrama caja pivote para un número de semillas de 30, en este diagrama ya se puede ver un efecto considerable en el incremento del largo de las grietas al incrementar el número de semillas, se puede observar que para la celda de 40x40 los mínimos y los máximos se vuelven más uniformes y llegan a un límite delimitado por el tamaño de la celda.

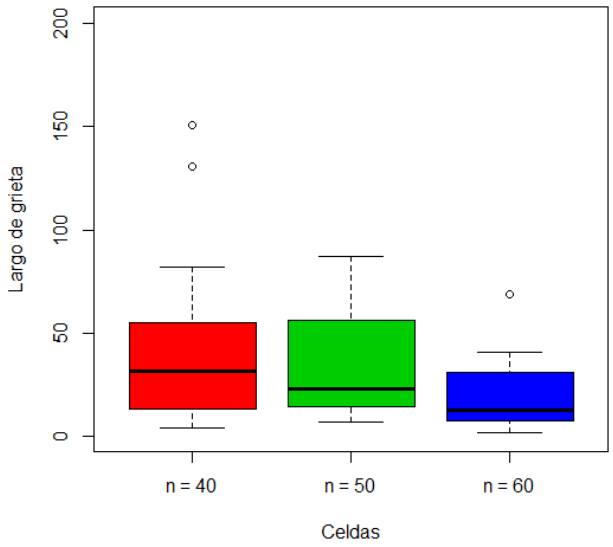

Fig. 3. Diagrama caja pivote para una k=30.

Conclusiones

De esta práctica se puede concluir la importancia de los diagramas de Voronoi en la todos las ramas de la ciencia, ya que con ellos se puede obtener información útil del comportamiento de un sistema en crecimiento o en constante cambio. Del experimento se puede concluir que tanto el tamaño de la celda como el número de semillas son importantes, ya que estas variables delimitan el crecimiento y propagación de la grieta.

Bibliografía

1) Sedgewick, Robert, (1995), *Algoritmos en C++*, Ed. Díaz de Santos.
2) Aurenhammer, Franz, (1991), *Voronoi Diagrams – A survey of a Fundamental Geometric Data Structure*, *ACM Computing* Surveys, 23(*3*), 345-405.