Reporte práctica 10

Algoritmo genético

A 17 de Octubre de 2017

Práctica

En esta práctica se analizó un algoritmo genético, se paralelizó y se estudió el efecto del paralelizado en el tiempo de ejecución del código original contra el código paralelizado.

Simulación y Resultados

Para la práctica se paralelizó el código original el cuál en función con el código de optimización del problema de la mochila permite obtener un máximo posible y la respuesta óptima a un problema, este en conjunto con el código original del algoritmo genético permiten obtener la mejor función factible presente en el algoritmo.

Para la paralelización se activaron las librerías presentes en el código, estas son la librería *doParallel* y la librería *foreach* las cuales permiten la paralelización del código.

```
1. suppressMessages(library(doParallel))
2. suppressMessages(library(foreach))
3. registerDoParallel(makeCluster(2))
```

A continuación se muestran las funciones realizadas para la aplicación de la paralelización.

```
1.  mu <- function(i){
2.   #for (i in 1:tam) { # cada objeto puede mutarse con probabilidad pm
   paralelizar
3.     if (runif(1) < pm) {
4.       return(mutacion(p[i,], n))}
5.   }
6.   muta <- foreach(i = 1:tam, combine = rbind) %dopar% mu(i)
7.   mutados <- data.frame(matrix(unlist(muta), ncol = n))
8.   colnames(mutados) <- c(1:n)
9.   colnames(p) <- c(1:n)
10.
11.        p <- rbind(p, mutados)
12.
13.        repro <- function(i){
14.        #for (i in 1:rep) { # una cantidad fija de reproducciones
   paralelizar
15.          padres <- sample(1:tam, 2, replace = FALSE)
16.          hijos <- reproduccion(p[padres[1],], p[padres[2],], n)
17.          p1 <- hijos[1:n] # primer hijo
18.          p2 <- hijos[(n+1):(2*n)] # segundo hijo
19.          jijos <- rbind(p1, p2)
20.          return(jijos)
21.        }
```

```
22.         p <- rbind(p, foreach(i = 1:rep,
   combine = rbind) %dopar% repro(i))
23.
24.         tam <- dim(p)[1]
25.         obj <- double()
26.         fact <- integer()
27.
28.         obj <- function(i){
29.         #for (i in 1:tam) { # paralelizar
30.           obj <- objetivo(p[i,], valores)
31.           fact <- factible(p[i,], pesos, capacidad)
32.           datos <- cbind(obj, fact)
33.           return(datos)
34.         }
35.
36.         rownames(p) <- c(1:dim(p)[1])
37.         p <- data.frame(sapply(p, function(x) as.numeric(as.character(x
   ))))
38.         p <- cbind(p, foreach(i = 1:tam,
   .combine = rbind) %dopar% obj(i))
39.         mantener <- order(-p[, (n + 2)], -p[, (n + 1)])[1:init]
40.         p <- p[mantener,]
41.         tam <- dim(p)[1]
42.         assert(tam == init)
43.         factibles <- p[p$fact == TRUE,]
44.         mejor <- max(factibles$obj)
45.         mejores <- c(mejores, mejor)
46.       }
47.       stopImplicitCluster()
```

Aquí se crearon varias funciones una para la parte que se encarga de generar las mutaciones, una para la parte de la reproducción y una para la parte de la selección de las soluciones factibles, cada una de ellas con su respectivo *foreach* para poder guardar los resultados y poder utilizarlos en el resultado final.

Para obtener un diagrama caja-bigotes del tiempo de ejecución de los códigos original y modificado contra el tiempo, se utilizó la función *sys.time()* para guardar el tiempo que dura la ejecución de los dos códigos y se usó el código siguiente:

```
1. codigo <- data.frame()
2. tiempo <- data.frame()
3.
4. for(i in 1:5){
5.   source('C:/Users/Pablo/Desktop/sim/p10/p10-4op.R', encoding = 'UTF-
   8')
6.   source('C:/Users/Pablo/Desktop/sim/p10/p10-4mod.R', encoding = 'UTF-
   8')
7.   tiempo <- cbind(total, totalor)
8.   codigo <- rbind(codigo, tiempo)
9. }
10.
11.       colnames(codigo) <- c("C. Original", "C. Modificado")
12.       boxplot(codigo, col = ("grey"), border = c("orange", "green"),
   ylab = "Tiempo (min)")
```

donde se utilizaron dos *data.frame* para guardar los resultados, un *for* para poder hacer varias ejecuciones de los códigos, los cuáles fueron añadidos al código con la función *source*, y para generar el diagrama caja-bigotes se utilizó el comando *boxplot*.

En la figura 1 se muestra el diagrama caja-bigotes del tiempo de ejecución de los códigos original y modificado, en éste se puede observar la diferencia de tiempo, donde el tiempo de ejecución para el código original es mayor que el tiempo del código modificado, esto puede deberse al tamaño de la población inicial.

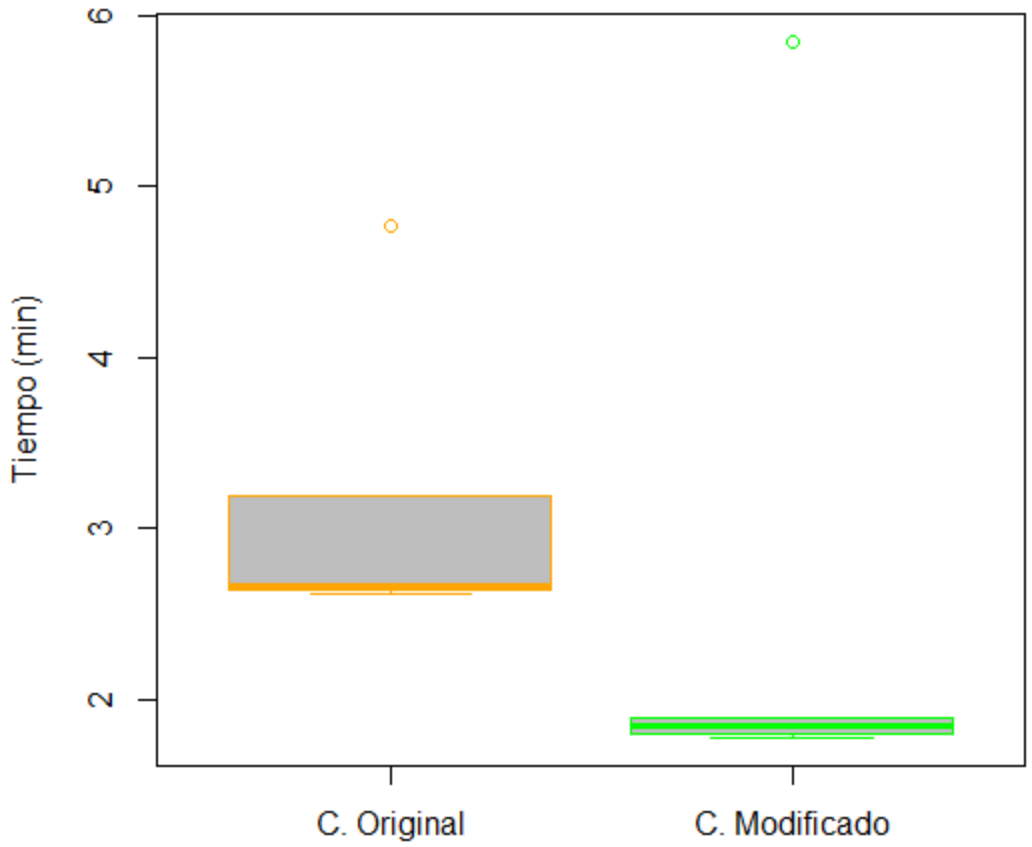

Fig. 1. Diagrama caja-bigotes de tiempo de ejecución de códigos original y modificado.

Para corroborar si el tamaño de la población inicial afecta el resultado se realizó el siguiente código:

```
1. datos <- data.frame()
2.
3. for(i in 1:3){
4.   for (init in seq(200, 300, 50)){
5.     source('C:/Users/Pablo/Desktop/sim/p10/p10-4op1.R',
   encoding = 'UTF-8')
6.     to <- cbind("Original", init, t)
7.     source('C:/Users/Pablo/Desktop/sim/p10/p10-4mod1.R',
   encoding = 'UTF-8')
```

```
8.      tm <- cbind("modificado", init, t)
9.      datos <- rbind(datos, to, tm)
10.         }
11.       }
12.
13.       save.image(file = "Datos.RData")
14.
15.       names(datos) <- c("Tipo","init","Tiempo")
16.       datos$init <- as.numeric(levels(datos$init))[datos$init]
17.       datos$Tiempo <- as.numeric(levels(datos$Tiempo))[datos$Tiempo]
18.       datos$Tipo <- as.factor(datos$Tipo)
19.       png("tiempos.png", width = 600, height = 800, pointsize = 15)
20.       boxplot(Tiempo ~ Tipo * init, data = datos, col = ("grey"),
    border = c("orange", "green"), xlab = "Poblaciones", ylab = "Tiempo
    (min)")
21.       legend("topleft", inset = 0.02, c("C. Original", "C.
    Modificado"), fill = c("orange", "green"), horiz=TRUE, cex=0.8,
    box.lty = 0)
22.       graphics.off()
```

En la figura 2 se muestra el diagrama caja-bigotes donde se puede apreciar que el tamaño de la población inicial es importante porque afecta directamente el tiempo de ejecución de los códigos, en el código secuencial sigue incrementando el tiempo de ejecución mientras que en el código modificado se puede ver la disminución del tiempo de ejecución en donde este tiempo es mayor para poblaciones de tamaños mayores pero en comparación con el código secuencial este tiempo sigue siendo mucho menor.

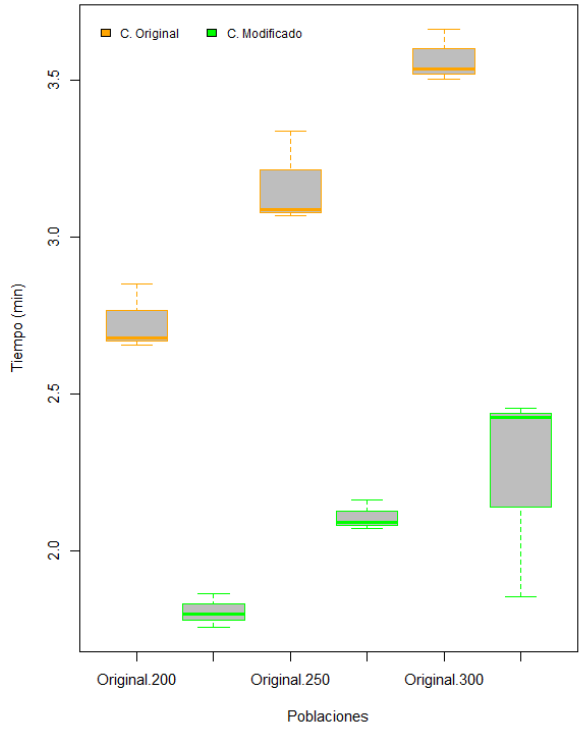

Fig. 2. Diagrama caja-bigotes de poblaciones contra el tiempo.

Conclusiones

En esta práctica se concluye que el algoritmo genético se puede mejorar al paralelizarlo, esto se nota en los tiempos de ejecución donde el tiempo de ejecución del código paralelizado será menor que el código secuencial.