Reporte práctica 10
Algoritmo genético

1. INTRODUCCIÓN

En la práctica diez se utiliza el problema de la mochila, donde la tarea consiste en seleccionar un subconjunto de objetos de tal forma que (i) no se exceda la capacidad de la mochila en términos de la suma de los pesos de los objetos incluidos, y que (ii) el valor total de los objetos incluidos sea lo máximo posible.

Mediante el problema de la mochila se implementó un algoritmo genético. Un algoritmo genético representa posibles soluciones a un problema en términos de un genoma que en nuestro caso va a ser un vector de verdades y falsos, indicando cuáles objetos vamos a incluir en la mochila (*TRUE* o 1 significa que llevamos el objeto, *FALSE* o 0 que lo descartamos de la selección).

El objetivo de la práctica es optimizar el tiempo de ejecución del código mediante la paralelización de las subrutinas que se encuentren en condiciones de ejecutarse en paralelo.

2. SIMULACIÓN

Se inicia la simulación con una población de 200 individuos, cada uno con 50 características o genes, en dónde cada gen tendrá un valor y un peso dado al azar.

Cada individuo puede *mutar* sus genes con una probabilidad *pm*. Después de obtener las mutaciones se eligen dos individuos (*padres*) para combinar sus genes y dar lugar a dos individuos (*hijos*). Todos los padres tienen la misma probabilidad de ser elegidos para la reproducción (recombinación).

Después de las cruzas se incrementará el número de individuos en la mochila por lo que será necesario elegir a sólo el número de la población inicial en base al acomodo por su valor, es decir quedarán los 200 individuos con mayor valor. Así se crea una nueva generación, y esto se repite *tmax* veces, el cual no da el número de generaciones en la simulación.

Para disminuir el tiempo de ejecución se tienen las rutinas en dónde se crean las mutaciones, las cruzas y cuando se eligen los individuos para las siguientes generaciones (*objetivos*, *factibles*).

Se reemplazan los *if* de estas rutinas por una función que se ejecutará con un llamado mediante *foreach* como se muestra en el código:

```
############################mutacion

 # for (i in 1: tam) {# cada objeto puede mutarse con probabilidad pm
  muta<-function (i) {
   if (runif (1) < pm) {
      return(mutacion(unlist(p[i,]), n))
    }
  }
  p<- rbind (p,foreach(i = 1:tam, .combine=rbind) %dopar% muta(i))
  ########################## cruzas
  #for (i in 1: rep) {# una cantidad fija de reproducciones

  cruza<- function(i) {
    padres <- sample (1: tam, 2, replace=FALSE)
    hijos <- reproduccion(p[padres [1],], p[padres[2],], n)
    hijo1 <-  hijos [1: n] # primer hijo
    hijo2 <-  hijos[(n+1): (2*n)] # segundo hijo
    hijos<-rbind (hijo1, hijo2)
    return(hijos)
   }
   p<- rbind (p,foreach(i = 1:rep, .combine=rbind) %dopar% cruza(i))
  ##########################objetivo factible
 # for (i in 1: tam) {#objetivo, factible
  fc<-function(i) {
    obj <- c(objetivo(p[i,], valores))
    fact <- c(factible(p[i,], pesos, capacidad))
    res<-(cbind(obj,fact))
    return(res)
     }
  p<-cbind (p,foreach(i=1:tam, .combine=rbind) %dopar% fc(i))
```

Para determinar la diferencia en ejecución de cada código (original y paralelo) se utilizó la herramienta *source* así mediante un *for* se determinaron cinco corridas y utilizando otro *for* se varió la población inicial secuencialmente en cien, doscientos y trecientos.

Se observó una diferencia notable entre el tiempo de ejecución del programa secuencial y el paralelo (figura 1). Se encontró que a mayor población inicial la diferencia en tiempo de ejecución entre el programa original y el paralelizado es mayor.

3. RESULTADOS

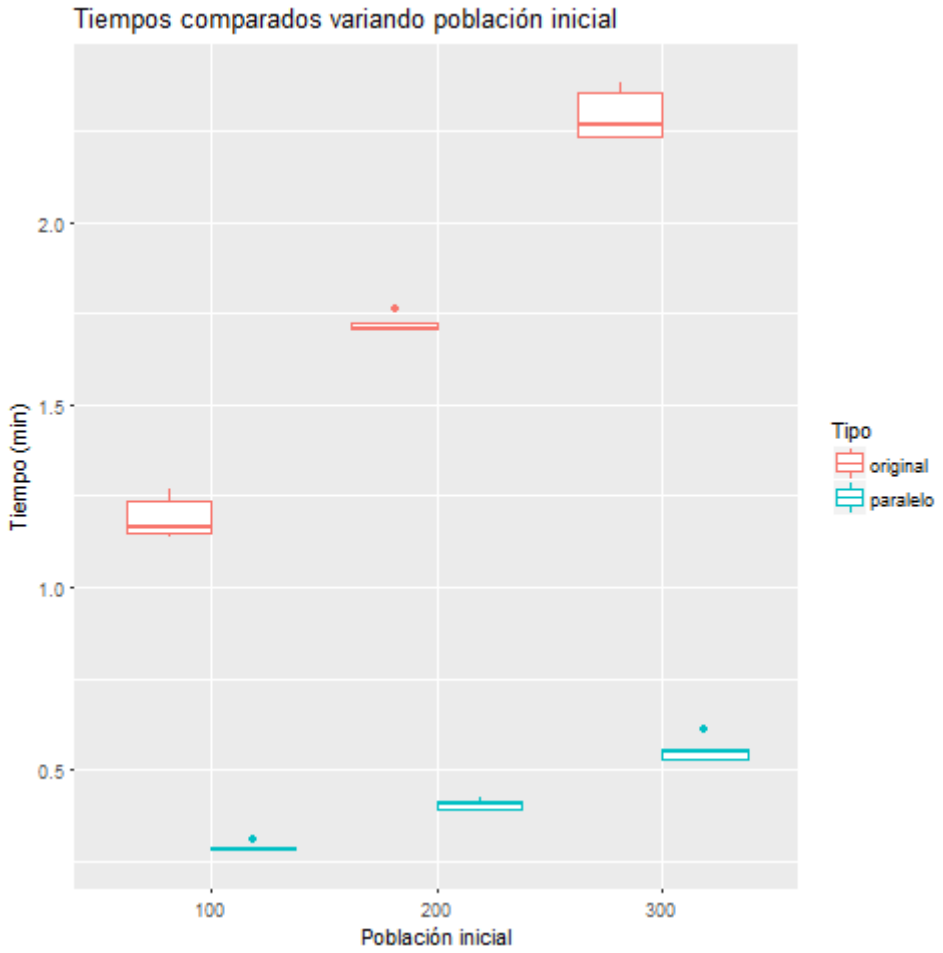

Figura 1. Gráfica caja-bigote de los tiempos de ejecución de los programas original y paralelizado variando la población inicial.

4. RETOS

- Reto 1

En el reto uno se pide que se modifique la manera en seleccionar los individuos que serán los padres en la reproducción, esto mediante la adición de una probabilidad que será directamente proporcional a su valor genético.

En el código original se utiliza la función *sample* para elegir a los padres al azar, pero para cumplir este reto se utilizó una herramienta *prob* de la función.

Se determinó la probabilidad en base a sus valores, primero se creó un vector de manera paralela, que contuviera los valores para cada individuo (*padresvalor*), posteriormente se guardó la división de cada valor entre la suma de todos los valores en el vector *pr*.

Se agregó una función *cruzaruleta* para implementar la probabilidad *pr* en la selección de los padres para la *cruza*.

En el código se muestran los cambios realizados para el cumplimiento del reto uno:

```
cruzaruleta<- function(i) {
    padres <- sample (1: tam, 2, prob = pr, replace=FALSE)
    hijos <- reproducción (p [padres [1],], p[padres [2],], n)
    hijo1 <-  hijos [1: n] # primer hijo
    hijo2 <-  hijos[(n+1): (2*n)] # segundo hijo
    hijos<-rbind (hijo1, hijo2)
    return(hijos)
  }
padresvalor<-foreach (i=1: tam, .combine = c)%dopar% objetivo(p[i,],
valores)
    pr= padresvalor/sum(padresvalor)
    p<- rbind (p,foreach(i = 1:rep, .combine=rbind) %dopar%
cruzaruleta(i)) #cruces con ruleta
```

Para observar visualmente las diferencias en las soluciones de ambos métodos de selección de los padres se utilizó un *for* para correr en el mismo programa ambos métodos.

En el siguiente código se muestran las dos subrutinas de los métodos de selección:

```
for (replica in 1:1){
  ############################con ruleta
  p <- poblacion.inicial(n, init)
  tam <- dim(p)[1]
  assert(tam == init)
  pm <- 0.05
  rep <- 50
  tmax <- 50
  mejoresrul <- double()

  for (iter in 1:tmax) {
    p$obj <- NULL
    p$fact <- NULL
    p<- rbind(p,foreach(i = 1:tam, .combine=rbind) %dopar% muta(i))
    padresvalor<-foreach(i=1:tam, .combine = c)%dopar% objetivo(p[i,],
valores)
    pr= padresvalor/sum(padresvalor)
    p<- rbind(p,foreach(i = 1:rep, .combine=rbind) %dopar%
cruzaruleta(i)) #cruces con ruleta
    tam <- dim(p)[1]
    obj <- double()
    fact <- integer()
    rownames(p)<-c(1:dim(p)[1])
```

```
    p<-data.frame
(sapply(p,function(x)as.numeric(as.character(x))))#numerico
    p<-cbind (p,foreach(i=1:tam, .combine=rbind) %dopar% fc(i))
#objetivo factible
    mantener <- order(-p[, (n + 2)], -p[, (n + 1)])[1:init]
    p <- p[mantener,]
    tam <- dim(p)[1]
    assert(tam == init)
    factibles <- p[p$fact == TRUE,]
    mejorul <- max(factibles$obj)
    mejoresrul <- c(mejoresrul, mejorul)
  }
  ################################sin ruleta
  p <- poblacion.inicial(n, init)
  tam <- dim(p)[1]
  assert(tam == init)
  pm <- 0.05
  rep <- 50
  tmax <- 50
  mejores <- double()

  for (iter in 1:tmax) {
    p$obj <- NULL
    p$fact <- NULL
      p<- rbind(p,foreach(i = 1:tam, .combine=rbind) %dopar% muta(i))
#mutaciones
      p<- rbind(p,foreach(i = 1:rep, .combine=rbind) %dopar%
cruza(i))#Cruces sin ruleta
    tam <- dim(p)[1]
    obj <- double()
    fact <- integer()
    rownames(p)<-c(1:dim(p)[1])
    p<-data.frame(sapply(p,function(x)as.numeric(as.character(x))))
    p<-cbind(p,foreach(i=1:tam, .combine=rbind) %dopar% fc(i))
#objetivo factible
    mantener <- order(-p[, (n + 2)], -p[, (n + 1)])[1:init]
    p <- p[mantener,]
    tam <- dim(p)[1]
    assert(tam == init)
    factibles <- p[p$fact == TRUE,]
    mejor <- max(factibles$obj)
    mejores <- c(mejores, mejor)
```

```
  }
    stopImplicitCluster()
  }
```

Finalmente se graficó mediante la librería *ggplot2*, para esto se guardaron los datos en un *data frame* para poder llamarlos al hacer las gráficas. En el código siguiente podemos ver que al graficarse por capas en *ggplot2* hace más fácil adicionar los dos métodos en la gráfica resultante.

```
g<-data.frame()

g<-cbind(1:tmax,mejores, mejoresrul)
colnames(g)=c("Paso", "Original", "Ruleta")
g<-as.data.frame(g)
  png(paste("P10R1", replica,".png"), width=600, height=300)
  library(ggplot2)
g1<-ggplot() +
  geom_line(data = g, aes(x=Paso , y=Original, color="Original"))+
    scale_y_continuous(name="Mayor valor", limits =c(
0.95*min(g$Original), 1.05*optimo)) +
  scale_x_continuous(name="Paso", limits = c(0, 50))+
  geom_hline(yintercept=optimo, linetype="dashed", color =
"green",size=0.6)

g1+geom_line(data = g, aes(x=g$Paso, y=g$Ruleta, color="Ruleta"))+
  guides(size=FALSE,color=guide_legend(title="Cruza"))
graphics.off()
```

Los resultados obtenidos para las soluciones con ambos métodos de selección de padres se muestran en la figura 2, dónde se puede observar una menor diferencia entre el óptimo y el mejor obtenido por el método ruleta que en el obtenido por padres elegidos al azar. Aunque la diferencia en la calidad de las soluciones no se observa drásticamente, podemos ver que el método ruleta siempre está por encima del método al azar.

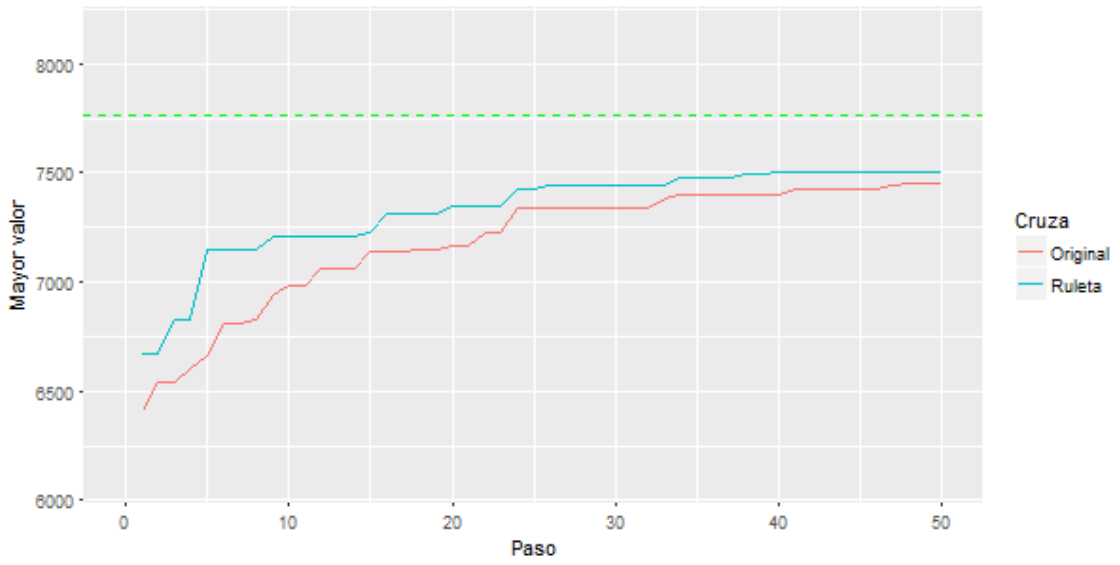

Figura 2. Gráfica de líneas de ambos métodos de selección de padres para la cruza (ruleta y original).

5. CONCLUSIONES

Se logró paralelizar el algoritmo genético mediante el uso de la herramienta *doparallel*.

Resulta un menor tiempo de ejecución al paralelizar las subrutinas *muta*, *cruza* y *fc*.

La diferencia en tiempos de ejecución del programa original al paralelizado es directamente proporcional al tamaño de la población inicial.

Se modificó el método de selección de los padres mediante la adición de la probabilidad *pr* en la función *cruzaruleta* con la herramienta *prob* de *sample*.

La probabilidad de selección es proporcional al valor de cada individuo, eso mediante la división de su valor entre la suma de todos los valores.

La selección de ruleta se muestra más cercana al valor óptimo en todas las generaciones.

La diferencia entre las soluciones obtenidas mediante los dos diferentes métodos, se vuelve menor a mayor número de generaciones.