Reporte práctica 11

Frentes de Pareto

A 24 de Octubre de 2017

Práctica

En esta práctica se analizó un generador de polinomios, el cual utiliza una optimización basada en objetivos multicriterio, donde el código original se paralelizó, se compararon los tiempos de ejecución del código original y el código modificado y se graficó el porcentaje de soluciones de Pareto como función del número funciones objetivo.

Simulación y Resultados

En la práctica se paralelizó el código original, solo se paralelizaron las partes donde convenía.

Para la paralelización se activaron las librerías *doParallel* y la librería *foreach* las cuales permiten la paralelización del código, y se creó un *cluster* con dos núcleos para la ejecución del código.

```
suppressMessages(library(doParallel))
suppressMessages(library(foreach))
registerDoParallel(makeCluster(2))
```

A continuación se muestran las funciones realizadas para la aplicación de la paralelización de las funciones objetivo y la evaluación de las soluciones.

```
obj <- function(i){
  return(poli(md, vc, tc))
}
obj <- foreach(i = 1:k, .combine = ) %dopar% obj(i)

minim <- (runif(k) > 0.5)
sign <- (1 + -2 * minim)
#n <- 200 # cuantas soluciones aleatorias
sol <- matrix(runif(vc * n), nrow=n, ncol=vc)
      val <- matrix(rep(NA, k * n), nrow=n, ncol=k)
      #################
      val<- function(i){
        p1 <- double()
        for (j in 1:k) { # para todos los objetivos
          p <- eval(obj[[j]], sol[i,], tc)
          p1<- cbind(p1, p)
        }
        return(p1)
      }
      val <- foreach(i = 1:n, .combine = rbind) %dopar% val(i)
```

En esta parte del código se crearon las funciones *obj* y *val* para realizar los cálculos y la evaluación de las funciones objetivo y las soluciones, cada una de estas con su respectivo *foreach* para poder guardar los resultados y poder utilizarlos en el siguiente fragmento del código.

```
fp <- function(i) {
#for (i in 1:n) {
  d <- logical()
  for (j in 1:n) {
    d <- c(d, domin.by(sign * val[i,], sign * val[j,], k))
  }
  cuantos <- sum(d)
  #dominadores <- c(dominadores, cuantos)
  #no.dom <- c(no.dom, cuantos == 0) # nadie le domina
      }
      dominadores <- foreach(i = 1:n, .combine = rbind) %dopar% fp(i)
```

En esta parte del código se añadió una tercera función *fp* para calcular los frentes de Pareto, este también tiene su propio *foreach* para guardar sus datos y utilizarlos en su posterior visualización.

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:

```
datos <- data.frame()

for(i in 1:3){
  for (k in c(5, 10, 15)) {
    source('C:/Users/Pablo/Desktop/sim/p11/tiempo/p11-5.R',
encoding = 'UTF-8')
    to <- cbind("O", k, t)
    source('C:/Users/Pablo/Desktop/sim/p11/tiempo/p11-5mod.R',
encoding = 'UTF-8')
    tm <- cbind("M", k, t)
    datos <- rbind(datos, to, tm)
      }
  }

  save.image(file = "Datos.RData")

  names(datos) <- c("Tipo","k", "Tiempo")
  datos$k <- as.numeric(levels(datos$k))[datos$k]
  datos$Tiempo <- as.numeric(levels(datos$Tiempo))[datos$Tiempo]
  datos$Tipo <- as.factor(datos$Tipo)
  png("tiempos.png", width = 600, height = 800, pointsize = 15)
  boxplot(Tiempo ~ Tipo * k, data = datos, col = ("grey"),
border = c("orange", "green"), xlab = "Funciones objetivo",
ylab = "Tiempo (min)")
  legend("topleft", inset = 0.02, c("C. Original", "C.
Modificado"), fill = c("orange", "green"), horiz=TRUE, cex=0.8,
box.lty = 0)
  graphics.off()
```

En 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 del tiempo de ejecución para distintos valores de *k*, estos se usaron para distinguir cuando es significativo el cambio del tiempo de ejecución en función de las funciones objetivo. Al tener funciones objetivo menores a 10 el tiempo de ejecución para el código original es mayor que el tiempo del código modificado, pero cuando las funciones objetivo son mayores o iguales a 10 el cambio en el tiempo de ejecución se vuelve significativo al disminuir considerablemente el tiempo del código modificado.

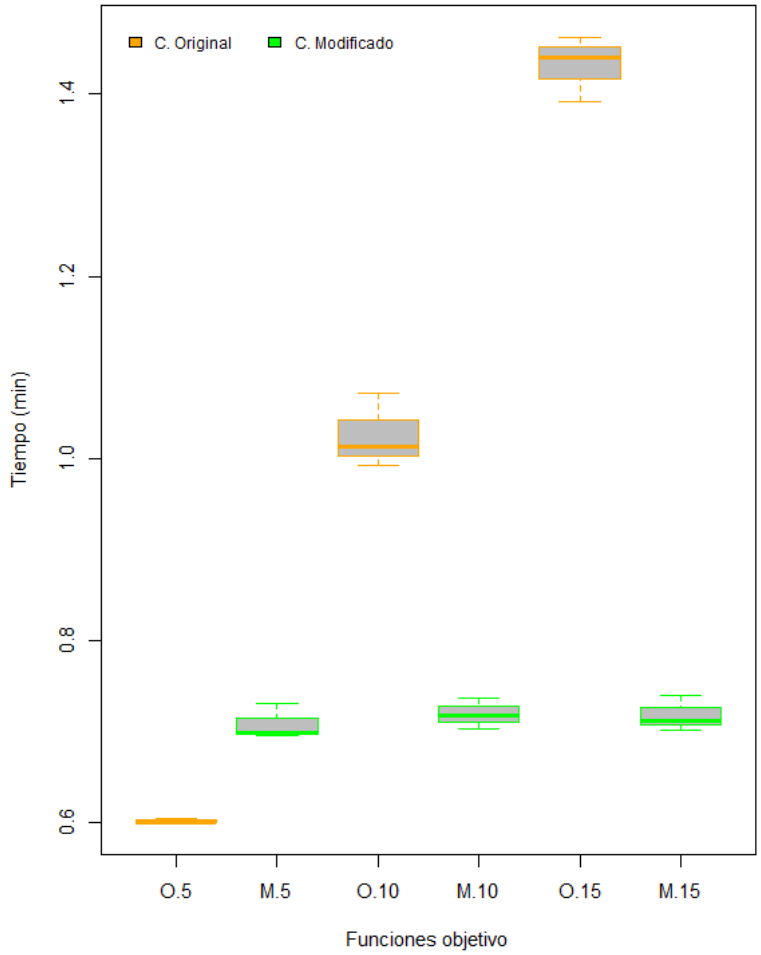

Fig. 1. Diagrama caja-bigotes de tiempo de ejecución de códigos original y modificado.

Para graficar el porcentaje de soluciones de Pareto como función del número funciones objetivo se utilizó el siguiente código.

```
suppressMessages(library(doParallel))
suppressMessages(library(foreach))
registerDoParallel(makeCluster(2))
tm <- data.frame()
resultados <- data.frame()

library(ggplot2)

for (i in 1:30){
        for (k in seq(2, 10, 2)){
          for (n in c(200)){
            source('C:/Users/Pablo/Desktop/sim/p11/Tarea/Tarea.R',
   encoding = 'UTF-8')
            tm <- cbind(dim(frente)[1], k, n)
            resultados <- rbind(resultados, tm)
          }
        }
      }
      stopImplicitCluster()

      names(resultados)=c("Dominadores", "Objetivos", "Soluciones")
      resultados$Objetivos <- as.factor(resultados$Objetivos)
      #resultados$Soluciones <- as.factor(resultados$Soluciones)

      ggplot(data = resultados, aes(resultados$Objetivos,
   resultados$Dominadores/n)) +
        geom_violin(scale="width",fill="yellow2", color="steelblue1")+
        geom_boxplot(width=0.2, fill="gray22", color="darkorchid1")+
        xlab("N◆mero de funciones objetivo k") +
        ylab("Soluciones %")+
        ggtitle("Cantidad de soluciones dominantes")
      ggsave(file=paste("p11_violin_n200r130.png", sep=""))
```

En este se activaron los paquetes *doParallel* y *foreach* para la ejecución rápida del código donde se varió el número de funciones objetivo en función del número de soluciones.

En la figura 2 se muestran los diagramas de violín para diferentes funciones objetivo en función del porcentaje del número de soluciones. En esta figura se puede visualizar como el número de funciones objetivo es un factor importante, ya que de éste depende el comportamiento de los diagramas de violín combinados con diagramas caja-bigotes. Al observar el diagrama de violín para una función objetivo de 2, la densidad de soluciones es menor ya que se tiene una menor cantidad de soluciones dominantes para la función objetivo por lo cual existen menos probabilidades de encontrar una solución óptima. Al incrementar el número de funciones objetivo la densidad de soluciones incrementa ya que existe una mayor probabilidad de que una solución sea dominante y perteneciente al frente de Pareto, esto se puede apreciar al incrementar las funciones objetivo.

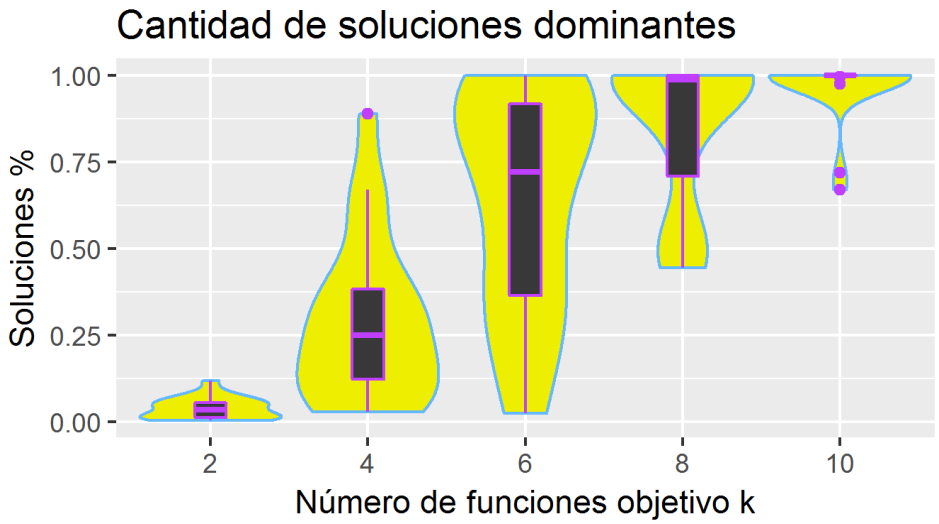

Fig. 2. Diagramas de violín del número de funciones objetivo contra el porcentaje de soluciones (200 soluciones).

De esta manera se puede inferir que el número de funciones objetivo es una variable que permite encontrar una mayor cantidad de soluciones a un problema dado ya que de esta manera existe una mayor adaptabilidad para la optimización multicriterio.

Conclusiones

Se puede concluir que la paralelización del código ayuda a disminuir los tiempos de ejecución del mismo, sobre todo al incrementar el número de funciones objetivo, y para obtener un cambio significativo en los tiempos de ejecución el número de funciones objetivo debe ser mayor o igual a 10. En el caso de las soluciones de Pareto el utilizarlas ayuda a la optimización multicriterio debido al incremento de funciones objetivo se puede incrementar la densidad de encontrar una solución óptima.

Reto 1

El primer reto consiste en elegir un subconjunto del frente de Pareto para diversificar las soluciones.

A continuación se muestra el código que se utilizó para realizar el reto 1.

```
no.dom <- logical()
for (i in 1:n){
  no.dom <- c(no.dom, dominadores[i] == 0)
}

frente <- subset(val, no.dom) # solamente las no dominadas
gf = dim(frente)[1]

png("Frente.png", pointsize = 16)
```

```
      par(mfrow = c(3, 1))
      plot(val[,1], val[,2], xlab = xl,
           ylab = yl)
      points(frente[,1], frente[,2], col="orange", pch=16, cex=1.5)


      if (gf > 2){
        frente<-as.data.frame(frente)
        colnames(frente)<-c("x","y")
        n.frente<-frente[order(frente$x),]

        distancia<-c()
        for (i in 1:gf-1){
          d<- sqrt((n.frente[i,]$x - n.frente[i+1,]$x)**2 + (n.frente[i
  ,]$y - n.frente[i+1,]$y)**2)
          distancia<-c(distancia, d)
        }
        dis <- mean(distancia)

        mantener <- rep(FALSE, gf)

        for (i in 1:gf){
          if (n.frente[i,] == head(n.frente,n=1)||n.frente[i,] == tail(
  n.frente,n=1)){
            mantener[i]=TRUE}else{
              j<-max(which(mantener))
              d<-
  sqrt((n.frente[i,]$x - n.frente[j,]$x)**2 + (n.frente[i,]$y-
  n.frente[j,]$y)**2)
              if(d >= dis){mantener[i]=TRUE}else{mantener[i]=FALSE}
            }
        }

        diversidad <- subset(n.frente, mantener)

        plot(val[,1], val[,2], xlab=xl,
             ylab=yl)
        points(frente[,1], frente[,2], col="orange", pch=16, cex=1.5)
        points(diversidad[,1], diversidad[,2], col="green", pch=16,
  cex=1.5)


        plot(val[,1], val[,2], xlab=xl,
             ylab=yl)
        points(diversidad[,1], diversidad[,2], col="green", pch=16,
  cex=1.5)

        graphics.off()

      }
```

En esta parte de código una vector *gf* para dimensionar el frente, posteriormente se guardó como un *data.frame* con la función *as.data.frame* y para ordenar los datos en el *frente* se utilizó el comando *colnames* para nombrar las columnas del *data.frame*, además se creó un vector *n.frente*

para ordenar las distancias en el eje *x*. Para calcular la distancia entre las soluciones del frente de Pareto se creó un vector vacío *distancia*, donde está en función de *i* de 1 a *gf-1*, a continuación se genera un vector *d* para calcular la distancia Euclidiana de los frentes, esto se guarda en el vector *distancia*, posteriormente se obtuvo el promedio de la distancia entre las soluciones del frente y se utilizó como umbral para la selección de las soluciones que se quedarán dentro del conjunto del frente de Pareto, esto se hace con el *for* en *i* de 1 a *gf* donde nuevamente se utilizó la distancia Euclidiana para seleccionar las soluciones que se quedan dentro del frente de Pareto. Con la función *subset* se seleccionan las filas y columnas deseadas para generar un nuevo *data.frame*. Por último se tiene las comandos para graficar la dispersión de las soluciones y el frente de Pareto con los comandos *plot* y *points*, y para unir las gráficas en un mismo documento se utilizó el comando *par(mfrow=c())*, las otras secciones del código se mantuvieron tal como las del código paralelizado.

En la figura 3 inciso a) se observa la gráfica de dispersión y su respectivo frente de Pareto el cual tiene una distribución muy cerrada, de este se diversificar las soluciones de ese frente de Pareto al buscar las soluciones más diversas las cuales se representan con las esferas de color verde, esto se puede ver en la figura 3 inciso b) donde además se observan las esferas naranjas del frente de Pareto para poder comparar la diversificación de las soluciones. En el inciso c) de la figura se tiene únicamente el frente de Pareto correspondiente al nuevo frente de Pareto el cual corresponde a un frente diversificado, esto quiere decir que no se tendrán soluciones similares donde no se pueda hacer la selección de una solución óptima, sino que se tendrá suficiente libertad para elegir una solución que se apegue a las necesidades o funciones objetivo deseadas.

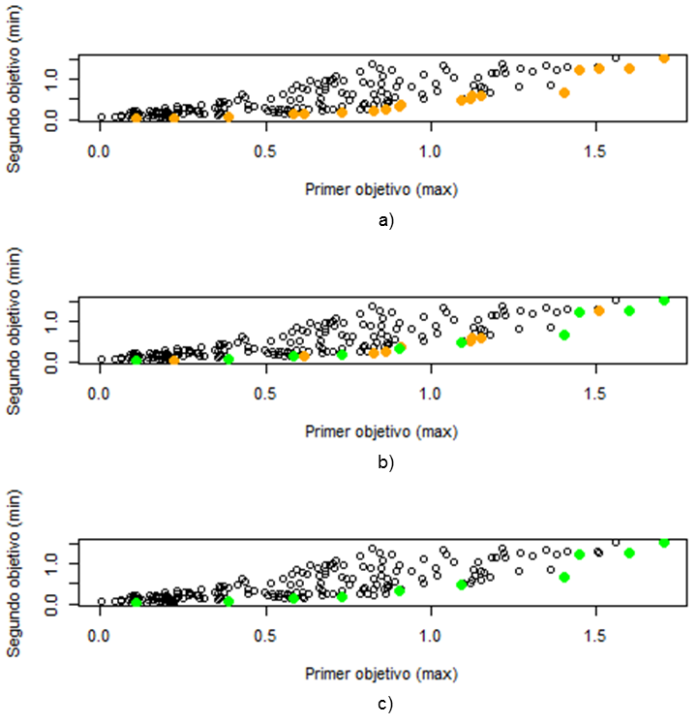

Fig. 3. Gráficos de dispersión, a) Frente de Pareto original, b) Frente de Pareto combinado, c) Frente de Pareto diversificado.

Conclusiones Reto 1

La diversificación de los frentes de Pareto permite analizar varias posibles soluciones que vayan dentro de las funciones objetivo, lo cual sirve para obtener la mejor solución posible sin tener que analizar todo un conjunto de datos o tener que seleccionar entre dos soluciones muy parecidas.