# Práctica 11: frentes de Pareto

Ricardo Rosas Macías

20 de mayo de 2019

## 1. Introducción

El análisis frentes de Pareto, permite obtener un resultado mediante la optimización de multi-objetivo, con ayuda de criterios de utilidad; lo cual permite discernir y proporcionar un solución en equilibrio de estas dos entidades, dentro de los márgenes de las variantes.

## 2. Objetivo

Se realizó cambios en el código del sitio web [3], de modo que proporcione una solución óptima que mejore en un objetivo, descartando eficazmente las demás opciones; de manera que esta sea dejada en la denominada frontera de Pareto.

### 2.1. Descripción

La finalidad del experimento es [3]:

> "Paralelizar el cálculo y graficar el porcentaje de soluciones de Pareto, como función del número de funciones objetivo con diagramas de violín combinados con diagramas de caja-bigote, verificando que las diferencias observadas sean estadísticamente significativas.
>
> El primer reto es seleccionar un subconjunto (cuyo tamaño como un porcentaje del frente original se proporciona como un parámetro) del frente de Pareto de tal forma que la selección esté diversificada, es decir, que no estén agrupados juntos en una sola zona del frente las soluciones seleccionadas.
>
> El segundo reto es adaptar el algoritmo genético de la práctica anterior para que vaya buscando mejora a un frente; la población inicial es el frente generado en la tarea y se aplique la diversificación del primer reto a cada generación después de los cruzamientos y las mutaciones."

# 3. Resultados y conclusiones

En base al trabajo anteriormente reportado [1], se realizó el código que se muestra en la parte inferior. En el cual las primeras líneas, exhiben los parámetros con lo que se ejecutó, asimismo en este se evidencia la paralelización de las treinta réplicas de ejecución para las funciones objetivo de 2–14.

```
library(parallel)
dat <- data.frame()
verify <- function(i){
  val <- c()
  for (j in 1:k) {
    val <- c(val, eval(obj[[j]], sol[i,], tc))
  }
  return(val)
}
prop <- function(i){
  return(list(poli(vc, md, tc)))
}
cluster <- makeCluster(detectCores()- 1)
clusterExport(cluster, c("domin.by", "sign", "eval", "poli", "pick.one"))

vc <- 4
md <- 3
tc <- 5
funciones <- seq(2, 14, by=1)
for (k in funciones) {
  for (replicas in 1:30) {
    obj <- list()
    clusterExport(cluster, c("vc", "md", "tc"))
    obj <- parSapply(cluster, 1:k, prop)
    minim <- (runif(k) > 0.5)
    sign <- (1 + -2 * minim)
    n <- 200 # soluciones aleatorias
    sol <- matrix(runif(vc * n), nrow=n, ncol=vc)
    val <- matrix(rep(NA, k * n), nrow=n, ncol=k)
    clusterExport(cluster, c("tc", "obj", "sol", "eval", "k", "n"))
    val <- parSapply(cluster, 1:n, verify)
    val <- t(val)

dat <- rbind(dat, c(k, replicas, sum(no.dom)/n))
    vec <- seq(1, k, by = 1)
    for (i in vec) {
      if((i+1) == is.element(i+1, vec)*(i+1)){
```

Con ayuda de la paquetería *ggplot2* [2] se obtuvo la visualización de código anterior. En la figura 1 evidencia un comportamiento creciente en las soluciones no dominadas dado al incrementar las funciones objetivo.

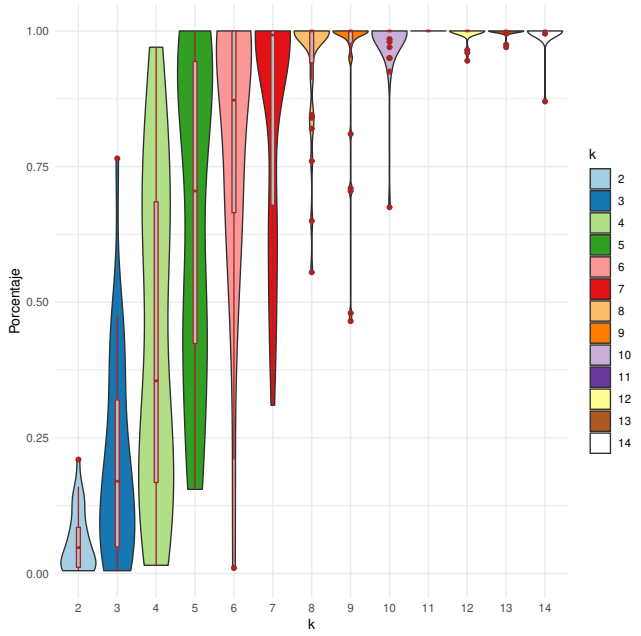



Figura 1: Porcentaje de soluciones

Asimismo, se realizó una prueba *Shapiro-Wilk* para verificar la probabilidad de una distribución normal; es una manera eficiente para determinar la normalidad de las variables. Por otro lado, se hizo una comprobación con *kruskal.test* para evidenciar la existencia de alguna diferencia entre los niveles, además se ejecutó una prueba *Pairwise Wilcox* para presentar que niveles son los causantes de estas diferencias.

```
a <- shapiro.test(dat$Porcentaje)
a$p.value
a$p.value < 0.05
b <- kruskal.test(dat$Porcentaje~dat$k)
b$p.value
b$p.value < 0.05
c <- pairwise.wilcox.test(dat$Porcentaje, dat$k, p.adjust.method = "none")
c$p.value
c$p.value < 0.05
```

Por consiguiente, se obtuvieron los resultados de *Shapiro-Wilk test* del código anterior, en donde se puede notar una diferencia en el valor $p < 0{,}05$, lo cual nos dice que hay diferencias relevantes en las cantidades de funciones objetivo. Por otra parte el valor $p < 0{,}05$ verificado con el *kruskal.test* nos muestra que solamente en algunos niveles las diferencias son despreciables. Por último con el *Pairwise Wilcox test* se expone con un *TRUE* a los niveles que presentan dichas diferencias.

```

a <- shapiro.test(dat$Porcentaje)
a$p.value
[1] 1.030554e-25
a$p.value < 0.05
```

```
[1] TRUE
b <- kruskal.test(dat$Porcentaje~dat$k)
b$p.value
[1] 7.036467e-50
b$p.value < 0.05
[1] TRUE

c <- pairwise.wilcox.test(dat$Porcentaje, dat$k, p.adjust.method = "none")
c$p.value

       2     3     4     5     6     7     8     9    10    11    12    13
3   TRUE    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
4   TRUE  TRUE    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
5   TRUE  TRUE  TRUE    NA    NA    NA    NA    NA    NA    NA    NA    NA
6   TRUE  TRUE  TRUE FALSE    NA    NA    NA    NA    NA    NA    NA    NA
7   TRUE  TRUE  TRUE FALSE FALSE    NA    NA    NA    NA    NA    NA    NA
8   TRUE  TRUE  TRUE  TRUE FALSE  TRUE    NA    NA    NA    NA    NA    NA
9   TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE    NA    NA    NA    NA    NA
10  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE    NA    NA    NA    NA
11  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE    NA    NA    NA
12  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
13  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE    NA
14  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
```

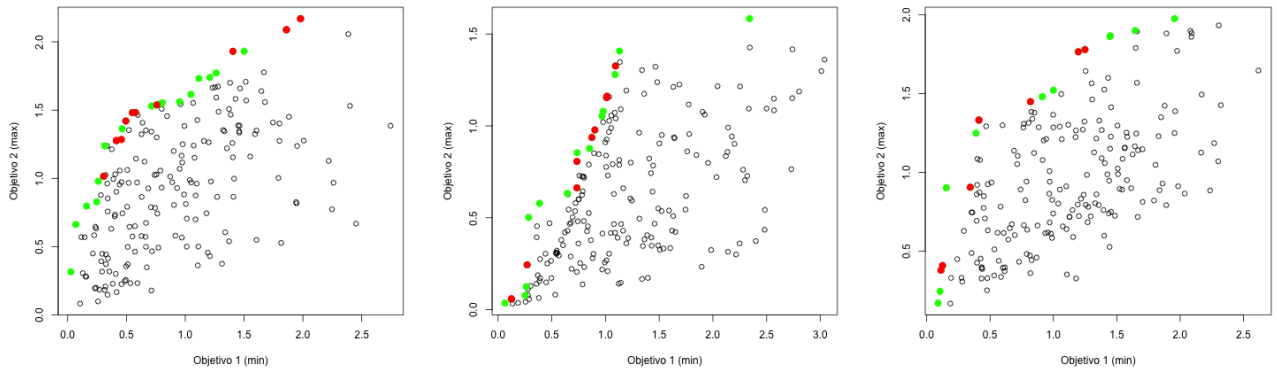
### 3.1. Reto 1

Para el primer reto se creó un código, de acuerdo al trabajo anteriormente reportado [1]. En el cual se seleccionó un subconjunto que permita observar el comportamiento del tamaño del porcentaje del frente de Pareto y este actué de modo diverso, ocasionando una agrupación en una sola zona, cercana a las soluciones.

```
cluster <- makeCluster(detectCores()- 1)
clusterExport(cluster, c("domin.by", "sign", "eval", "poli", "n", "pick.one"))

dat <- rbind(dat, c(k, replicas, sum(no.dom)/n))
    Select <- frente[c(round(runif(round(dim(frente)[1]/2), min = 1, max = dim(frente)
        [1]))),]
    Select1 <- data.frame()
    Select1 <- rbind(Select1, Select)

vec <- seq(1, k, by = 1)
    for (i in vec) {
      if((i+1) == is.element(i+1, vec)*(i+1)){
        png(paste("p11_frente", k, "_", replicas, "_", i, "-", i+1, ".png", sep=""))
        xt = paste("Objetivo ", i, " (", cual[minim[i] + 1], ")", sep = "")
        yt = paste("Objetivo ", i+1, " (", cual[minim[i+1] + 1], ")", sep = "")
        plot(val[,i], val[,i+1], xlab=xt, ylab=yt)
        points(frente[,i], frente[,i+1], col="green", pch=16, cex=1.5)
        points(Select1[,i], Select1[,i+1], col="red", pch=16, cex=1.5)
        graphics.off()
```

```
      }
    }
    data <- data.frame(pos=rep(0, n), dom=dominadores)
```

En la figura 2 se manifiesta un frente diversificado en atención a lo cual los puntos rojos son los puntos del frente de Pareto y los verdes al frente original. Se puede notar que al aumentar los pasos en la ejecución del código los puntos rojos van disminuyendo, debido a la selección determinada en el código anterior.




(a) Paso 1 (b) Paso 6 (c) Paso 9

Figura 2: Divergencia de variables

## 3.2. Reto 2

En el segundo reto se realizo una combinación del código del algoritmo genético [4] con el proporcionado para esta práctica, para realizar una especie de selección natural que permita tener una mejor generación de la población inicial, como se muestra en las siguientes líneas de código.

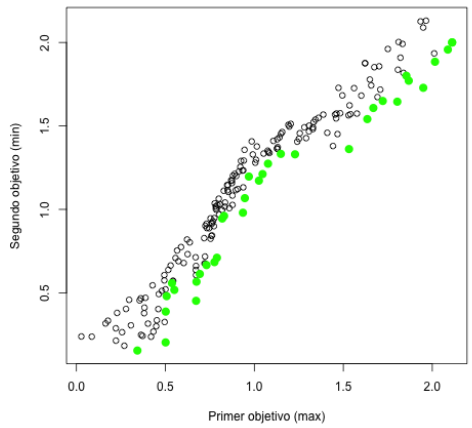
```
conquered <- function(i){
  d <- logical()
  for (j in 1:n) {
    d <- c(d, domin.by(sign * val[i,], sign * val[j,], k))
  }
  return(d)
}

cluster <- makeCluster(detectCores() - 1)
clusterExport(cluster, "verify")
clusterExport(cluster, "eval")
clusterExport(cluster, "obj")
clusterExport(cluster, "sol")
clusterExport(cluster, "tc")
clusterExport(cluster, "k")
clusterExport(cluster, "n")
```

```
17
18 val <- parSapply(cluster, 1:n, verify)
19 val <- t(val)
20 stopCluster(cluster)
21
22 mejor1 <- which.max(sign[1] * val[,1])
23 mejor2 <- which.max(sign[2] * val[,2])
24 cual <- c("max", "min")
25 xl <- paste("Primer objetivo (", cual[minim[1] + 1], ")", sep="")
26 yl <- paste("Segundo objetivo (", cual[minim[2] + 1], ")", sep="")
27
28 no.dom <- logical()
29 conqueredres <- integer()
30
31 cluster <- makeCluster(detectCores() - 1)
32 clusterExport(cluster, "conquered")
33 clusterExport(cluster, "domin.by")
34 clusterExport(cluster, "val")
35 clusterExport(cluster, "sign")
36 clusterExport(cluster, "k")
37 clusterExport(cluster, "n")
38
39 d <- parSapply(cluster, 1:n, conquered)
40 stopCluster(cluster)
41
42 for(x in 1:nrow(sol)){
43     cuantos <- sum(d[,x])
44     conqueredres <- c(conqueredres, cuantos)
45     no.dom <- c(no.dom, cuantos == 0)
46     dom <- c(dom, cuantos != 0)
47 }
48 frente <- subset(val, no.dom)
49
50 mutacion <- function(sol, vc) {
51   pos <- sample(1:vc, 1)
52   mut <- sol
53   delta <- 0.1
54   mut[pos] <- (sol[pos]) * delta
55   return(mut)
56 }
57
58 muta <- function(i){
59   if (runif(1) < pm) {
60     return(mutacion(sol[i,], vc))
61   }
62   else{
63     return(sol[i,])
64   }
65 }
66
67 reproduccion <- function(x, y, vc) {
68   pos <- sample(2:(vc-1), 1)
69   xy <- c(x[1:pos], y[(pos+1):vc])
70   yx <- c(y[1:pos], x[(pos+1):vc])
71   return(c(xy, yx))
```

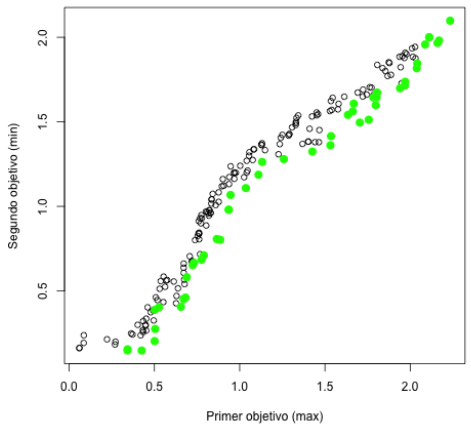
```
}


cluster <- makeCluster(detectCores() - 1)

pm <- 0.05
rep <- 50
tmax <- 100
for (iter in 1:tmax) {

clusterExport(cluster, "pm")
  clusterExport(cluster, "vc")
  clusterExport(cluster, "sol")
  clusterExport(cluster, "mutacion")
  clusterExport(cluster, "muta")
  sol <- t(parSapply(cluster, 1:n,muta))

  for (i in 1:rep) {
    padres <- sample(1:n, 2, replace=FALSE)
    hijos <- reproduccion(sol[padres[1],], sol[padres[2],], vc)
    sol <- rbind(sol, hijos[1:vc]) # primer hijo
    sol <- rbind(sol, hijos[(vc+1):(2*vc)]) # segundo hijo
  }
  val <- matrix(rep(NA, k * nrow(sol)), nrow=nrow(sol), ncol=k)
  clusterExport(cluster, "verify")
  clusterExport(cluster, "eval")
  clusterExport(cluster, "obj")
  clusterExport(cluster, "sol")
  clusterExport(cluster, "tc")
  clusterExport(cluster, "k")
  clusterExport(cluster, "n")
  val <- parSapply(cluster, 1:nrow(sol), verify)
  val <- t(val)
  no.dom <- logical()
  dom <- logical()
  conqueredres <- integer()
  clusterExport(cluster, "conquered")
  clusterExport(cluster, "domin.by")
  clusterExport(cluster, "val")
  clusterExport(cluster, "sign")
  clusterExport(cluster, "k")
  clusterExport(cluster, "n")

  d <- parSapply(cluster, 1:nrow(sol), conquered)
  for(x in 1:nrow(sol)){
    cuantos <- sum(d[,x])
    conqueredres <- c(conqueredres, cuantos)
    no.dom <- c(no.dom, cuantos == 0)
    dom <- c(dom, cuantos != 0)
  }
  frente_sol <- subset(sol, no.dom)
  dominadas <- subset(sol, dom)
  frente <- subset(val, no.dom)
  domi <- order(conqueredres)
  domi <- domi[1:n]
```

```
127
128   sol <- sol[domi,]
129   val <- val[domi,]
130   digitos <- floor(log(tmax, 10)) + 1
131   tl <- paste0(iter, "", sep="")
132   while (nchar(tl) < digitos) {
133     tl <- paste("0", tl, sep="")
134   }
135
136  if(nrow(frente) == n)
137   {
138     break;
139   }
140 }
141 stopCluster(cluster)
142 system("convert -delay 50 -size 300x300 Genetico*.png -loop 0 Gen.gif")
```

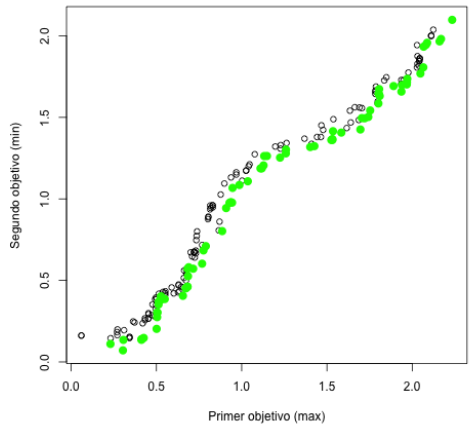
Fueron seleccionadas cinco representaciones gráficas significativas del cambio, que conforman a la figura 3, en la cual se muestra la distribución ocasionada por el algoritmo genético, que en virtud de ello el frente tiene cambios relevantes en las primeras generaciones, después de la octava generación las soluciones se van homogeneizando ligeramente.



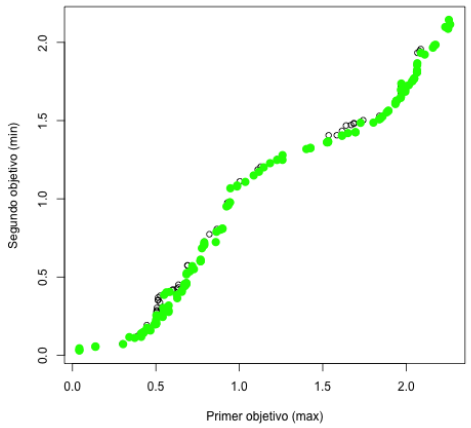

(a) Generación 1



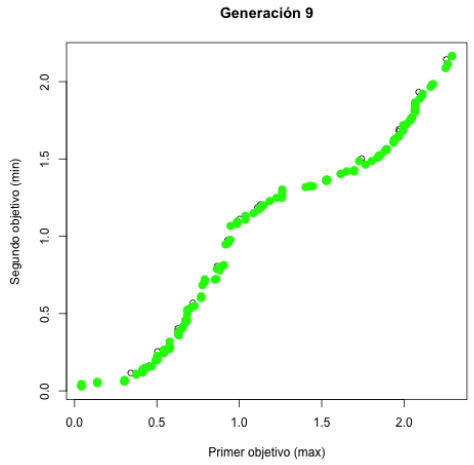

(b) Generación 3



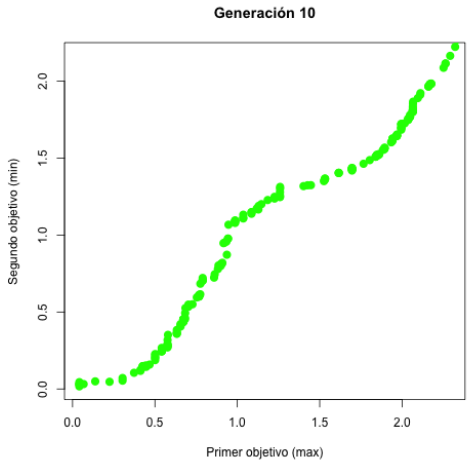

(c) Generación 5




(d) Generación 8




(e) Generación 9




(f) Generación 10

Figura 3: Distribución de algoritmo genético

# Referencias


[1] Yessica Reyna Fernández. Practica 11, 2018. URL https://sourceforge.net/projects/simulacion-de-sistemas/files/Practica%2011/.

[2] Alboukadel Kassambara. ggplot2 violin plot : Quick start guide - r software and data visualization, 2015. URL http://www.sthda.com/english/wiki/ggplot2-violin-plot-quick-start-guide-r-software-and-data-visualization.

[3] Elisa Schaeffer. Práctica 11: frentes de pareto, 2019. URL https://elisa.dyndns-web.com/teaching/comp/par/p11.html.

[4] Elisa Schaeffer. Práctica 10: algoritmo genético, 2019. URL https://elisa.dyndns-web.com/teaching/comp/par/p10.html.