TAREA # 10
Algoritmo genético

Natalia Berenice Pérez López

3 de noviembre de 2021

1. Objetivo

El objetivo de esta práctica es generar tres instancias con tres distintas reglas (nueve en total):

Regla 1: el peso y el valor de cada objeto se generan independientemente con una distribución uniforme.

Regla 2: el valor de cada objeto se genera independientemente con una distribución exponencial y su peso es inversamente correlacionado con el valor.

Regla 3: el peso de cada objeto se generan independientemente con una distribución normal y su valor es (positivamente) correlacionado con el cuadrado del peso, con un ruido normalmente distribuido de baja magnitud.

Determinar para cada uno de los tres casos si variar la probabilidad de mutación, la cantidad de cruzamientos y el tamaño de la población (dos o tres niveles basta) tienen un efecto (solo o en combinación) estadísticamente significativo (usando por lo menos tres réplicas por instancia) en la calidad de resultado, manteniendo el tiempo de ejecución fijo.

2. Desarrollo

Para generar los códigos de esta práctica se realizaron algunas ideas y pruebas iniciales, las cuales se encuentran en mi repositorio en GitHub. Se inició tomando como base el código revisado en clase para ejecutar una cantidad predeterminada de generaciones, primero mutando, luego reproduciendo, y al final cortando el tamaño de la población a la misma que estuvo al inicio de la iteración, dando preferencia a las soluciones factibles [2]. Se realizaron tres códigos diferentes para cada regla mencionada en el objetivo de la práctica, y para cada regla se realizaron tres códigos más para variar la probabilidad de mutación, la cantidad de cruzamientos y el tamaño de la población en cada código respectivamente. Para el código que mantiene el tiempo de ejecución fijo me apoyé en el repositorio de mi compañero Eduardo Navarro [1].

A continuación se muestran los códigos para cada regla:

REGLA 1: el peso y el valor de cada objeto se generan independientemente con una distribución uniforme.

```
#pesos independientes con distribucion uniforme
generador.pesos <- function(cuantos, min, max) {
  return(sort(round((runif(cuantos)) * (max - min) + min)))
}
#valores independientes con distribucion uniforme
generador.valores <- function(cuantos, min, max) {
  return(round((runif(cuantos)) * (max - min) + min))
}
```

Listing 1: Código para cumplir la regla 1.

REGLA 2: el valor de cada objeto se genera independientemente con una distribución exponencial y su peso es inversamente correlacionado con el valor.

```
#valores independientes con distribucion exponencial
generador.valores <- function(cuantos, min, max) {
  return(round(normalizar(rexp(cuantos)) * (max - min) + min))
```

```
}

#pesos inversamente correlacionados con el valor
generador.pesos <- function(valores, min, max) {
  n <- length(valores)
  pesos <- double()
  for (i in 1:n) {
    media <- valores[i]
    desv <- runif(1, max=0.1)
    pesos <- c(pesos, rnorm(1, (1/media), desv))
  }
  pesos <- normalizar(pesos) * (max - min) + min
  return(pesos)
}
```

Listing 2: Código para cumplir la regla 2.

REGLA 3: el peso de cada objeto se genera independientemente con una distribución normal y su valor es (positivamente) correlacionado con el cuadrado del peso, con un ruido normalmente distribuido de baja magnitud.

```
#pesos independientes con distribucion normal
generador.pesos <- function(cuantos, min, max) {
  return(sort(round(normalizar(rnorm(cuantos)) * (max - min) + min)))
}
#valores correlacionados con el cuadrado del peso con ruido normalmente distribuido de baja
    magnitud
generador.valores <- function(pesos, min, max) {
  n <- length(pesos)
  valores <- double()
  for (i in 1:n) {
    media <- pesos[i]
    desv <- runif(1)
    ruido <- rnorm(1, sd=.1)
    valores <- c(valores, rnorm(1, media^2, desv) + ruido)
  }
  valores <- normalizar(valores) * (max - min) + min
  return(valores)
}
```

Listing 3: Código para cumplir la regla 3.

A continuación se muestran los tres códigos que se ejecutaron con cada regla:

Código 1: variar la probabilidad de mutación en tres niveles (0.1, 0.3 y 0.6) y hacer tres réplicas.

```
df = data.frame()
mut = c(0.1, 0.3, 0.6) #variar probabilidad de mutacion
rep <- 10 #cantidad de cruzamientos

for (pm in mut){
  for (replica in 1:3){
    n <- 60 #cuantos objetos
    pesos <- generador.pesos(n, 15, 80)
    valores <- generador.valores(pesos, 10, 500)
    capacidad <- round(sum(pesos) * 0.65)
    optimo <- knapsack(capacidad, pesos, valores)
    init <- 30  #cuantas soluciones
    p <- poblacion.inicial(n, init)
    tam <- dim(p)[1]
    assert(tam == init)
    mejores <- double()

    tiempo = 10 #segundos
    start = Sys.time()

    while(TRUE) {
      elapsed = as.numeric(difftime(Sys.time(), start, units = 'secs'))
      remaining = tiempo - round(elapsed)
      Sys.sleep(0.1)
      print(remaining)

      p$obj <- NULL
```

```
      p$fact <- NULL
      for (i in 1:tam) { # cada objeto puede mutarse con probabilidad pm
        if (runif(1) < pm) {
          p <- rbind(p, mutacion(p[i,], n))
        }
      }
      for (i in 1:rep) { # una cantidad fija de reproducciones
        padres <- sample(1:tam, 2, replace=FALSE)
        hijos <- reproduccion(p[padres[1],], p[padres[2],], n)
        p <- rbind(p, hijos[1:n]) # primer hijo
        p <- rbind(p, hijos[(n+1):(2*n)]) # segundo hijo
      }
      tam <- dim(p)[1]
      obj <- double()
      fact <- integer()
      for (i in 1:tam) {
        obj <- c(obj, objetivo(p[i,], valores))
        fact <- c(fact, factible(p[i,], pesos, capacidad))
      }
      p <- cbind(p, obj)
      p <- cbind(p, fact)
      mantener <- order(-p[, (n + 2)], -p[, (n + 1)])[1:init]
      p <- p[mantener,]
      tam <- dim(p)[1]
      assert(tam == init)
      factibles <- p[p$fact == TRUE,]
      mejor <- max(factibles$obj)
      mejores <- c(mejores, mejor)

      print(paste(mejor, (optimo - mejor) / optimo))
      opt <- ((optimo - mejor) / optimo)*100
      segundos <-round(elapsed)

      if (remaining <= 0) break

      resultado = c(pm, replica, segundos, mejor, opt, optimo)
      df = rbind(df, resultado)
      names(df) = c("Mutacion", "Replica", "Segundo", "Mejor", "Gap", "Optimo")
    }
  }
}
```

Listing 4: Fragemento del código para variar la probabilidad de mutación en tres niveles.

En el código anterior se muestra el ciclo `While` para mantener un tiempo de ejecución fijo de 10 segundos en lugar de hacer iteraciones.

Código 2: variar la cantidad de cruzamientos en tres niveles (10, 15 y 20) y hacer tres réplicas.

```
df = data.frame()
cruz = c(10, 15, 20) #variar la cantidad de cruzamientos
pm = 0.1 #probabilidad de mutacion

for (rep in cruz){
  for (replica in 1:3){
    n <- 60 #cuantos objetos
    pesos <- generador.pesos(n, 15, 80)
    valores <- generador.valores(pesos, 10, 500)
    capacidad <- round(sum(pesos) * 0.65)
    optimo <- knapsack(capacidad, pesos, valores)
    init <- 30  #cuantas soluciones
    p <- poblacion.inicial(n, init)
    tam <- dim(p)[1]
    assert(tam == init)
    mejores <- double()

    tiempo = 10 #segundos
    start = Sys.time()

    while(TRUE) {
      elapsed = as.numeric(difftime(Sys.time(), start, units = 'secs'))
```

```
      remaining = tiempo - round(elapsed)
      Sys.sleep(0.1)
      print(remaining)
```

Listing 5: Fragemento del código para variar la cantidad de cruzamientos en tres niveles.

Código 3: variar el tamaño de la población en tres niveles (25, 35 y 45) y hacer tres réplicas.

```
df = data.frame()
pm = 0.1 #probabilidad de mutacion
rep <- 15 #cantidad de cruzamientos
solu = c(25, 35, 45) #cuantas soluciones

for (init in solu){
  for (replica in 1:3){
    n <- 60 #cuantos objetos
    pesos <- generador.pesos(n, 15, 80)
    valores <- generador.valores(pesos, 10, 500)
    capacidad <- round(sum(pesos) * 0.65)
    optimo <- knapsack(capacidad, pesos, valores)
    p <- poblacion.inicial(n, init)
    tam <- dim(p)[1]
    assert(tam == init)
    mejores <- double()

    tiempo = 10 #segundos
    start = Sys.time()

    while(TRUE) {
      elapsed = as.numeric(difftime(Sys.time(), start, units = 'secs'))
      remaining = tiempo - round(elapsed)
      Sys.sleep(0.1)
      print(remaining)
```

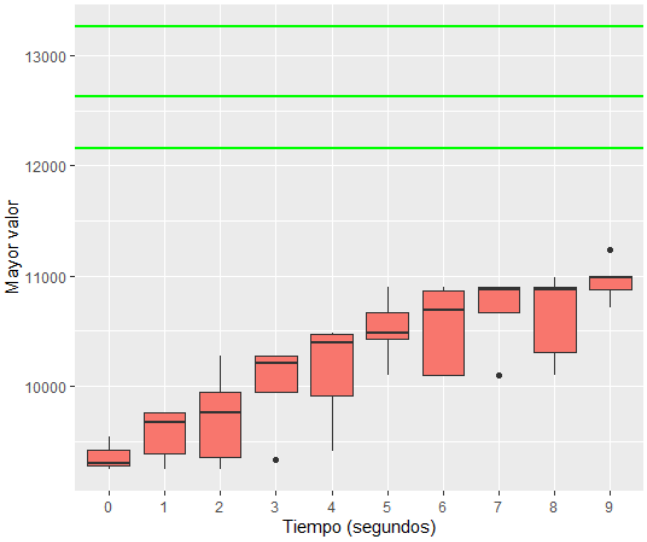
Listing 6: Fragemento del código para variar el tamaño de la población en tres niveles.

En la figura 1 se muestran los diagramas caja-bigote para cada probabilidad de mutación del `código 1` de la `regla 1`.
En la figura 2 se muestran los diagramas caja-bigote para cada cantidad de cruzamientos del `código 2` de la `regla 1`.
En la figura 3 se muestran los diagramas caja-bigote para cada tamaño de población del `código 3` de la `regla 1`.

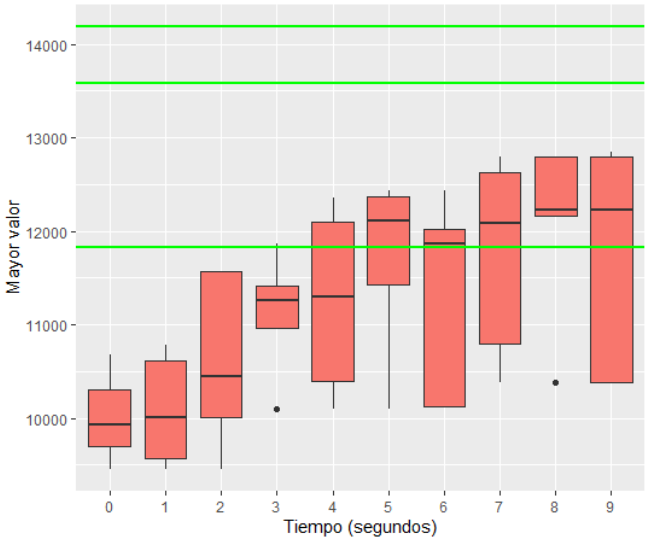
En la figura 4 se muestran los diagramas caja-bigote para cada probabilidad de mutación del `código 1` de la `regla 2`.
En la figura 5 se muestran los diagramas caja-bigote para cada cantidad de cruzamientos del `código 2` de la `regla 2`.
En la figura 6 se muestran los diagramas caja-bigote para cada tamaño de población del `código 3` de la `regla 2`.

En la figura 7 se muestran los diagramas caja-bigote para cada probabilidad de mutación del `código 1` de la `regla 3`.
En la figura 8 se muestran los diagramas caja-bigote para cada cantidad de cruzamientos del `código 2` de la `regla 3`.
En la figura 9 se muestran los diagramas caja-bigote para cada tamaño de población del `código 3` de la `regla 3`.

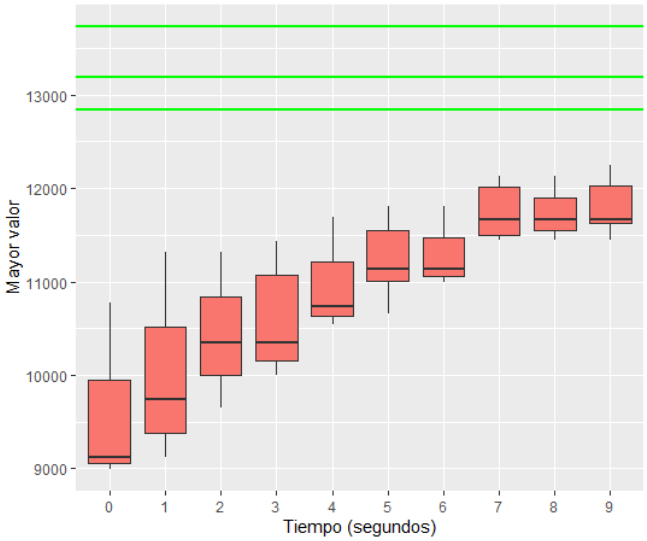
Para analizar mejor los resultados de los diagramas caja-bigote se realizaron pruebas estadísticas, las cuales se muestran después de cada figura.



(a) Probabilidad de mutación = 0.1



(b) Probabilidad de mutación = 0.3



(c) Probabilidad de mutación = 0.6

Figura 1: Mayor valor para cada tiempo de acuerdo a cada probabilidad de mutación para el código 1 de la regla 1.

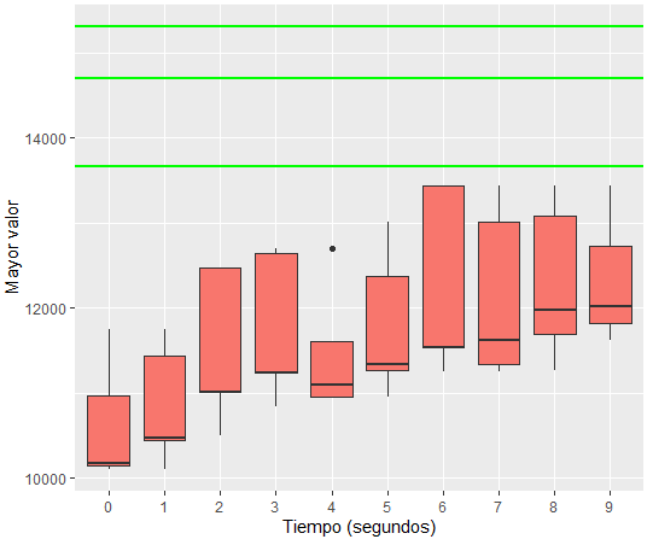
Cuadro 1: Resultados de la prueba estadística.

Outliers	0
Normalidad por grupo	0,1: $p = 0{,}00348$ / 0,3: $p = 0{,}00139$ / 0,6: $p = 0{,}00440$
Kruskal Wallis	$p = 8{,}095 \times 10^{-6}$

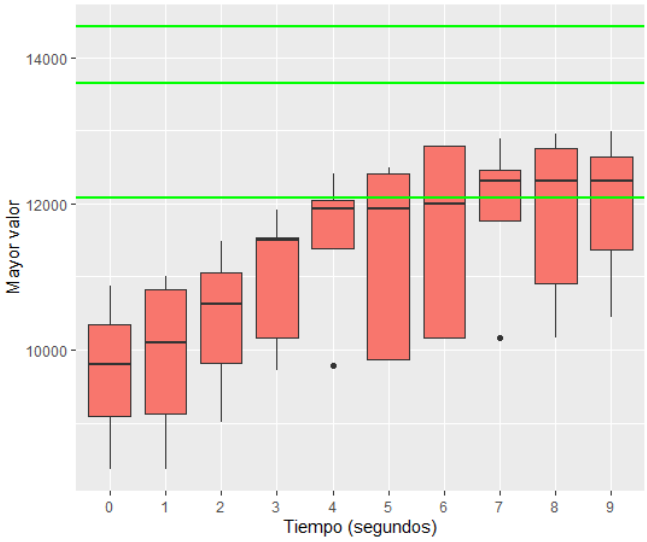
Cuadro 2: Resultados al aplicar la prueba `Wilcoxon`.

Valor de p	0,1	0,3
0,3	0,00015	-
0,6	0,00004	0,29082

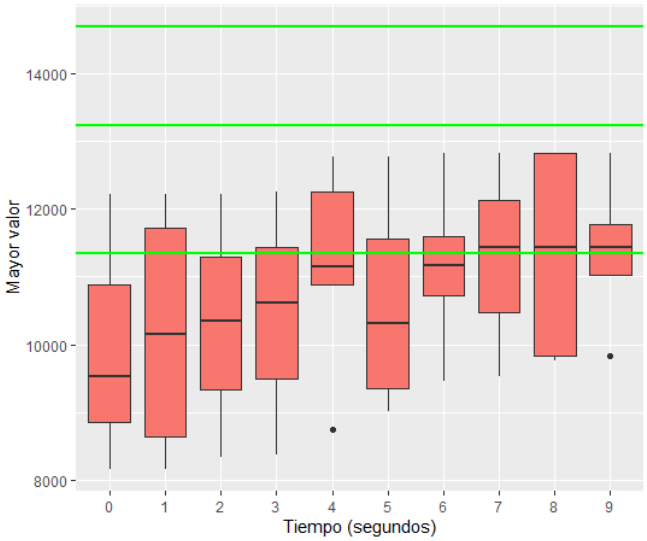
Hipótesis nula : Las medias son iguales en todos los grupos. Se rechaza esta hipótesis, si hay diferencias significativas de las medias.



(a) Cruzamientos = 10



(b) Cruzamientos = 15



(c) Cruzamientos = 20

Figura 2: Mayor valor para cada tiempo de acuerdo a cada cantidad de cruzamientos para el código 2 de la regla 1.

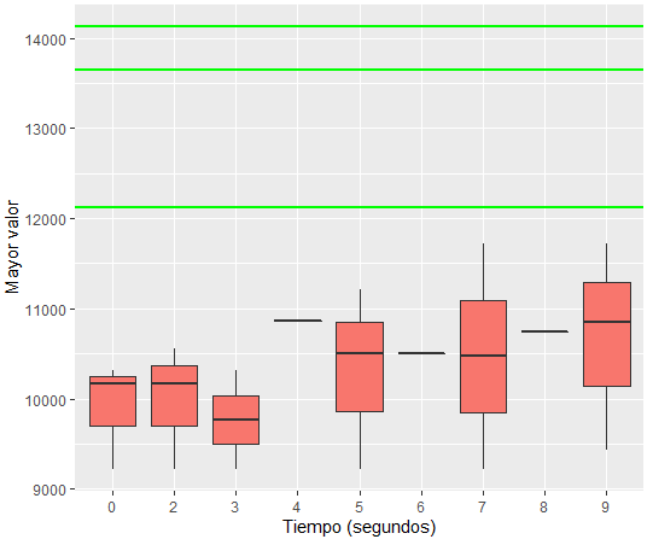
Cuadro 3: Resultados de la prueba estadística.

Outliers	0
Normalidad por grupo	10: $p = 0{,}00571$ / 15: $p = 0{,}01540$ / 20: $p = 0{,}00831$
Kruskal Wallis	$p = 0{,}01654$

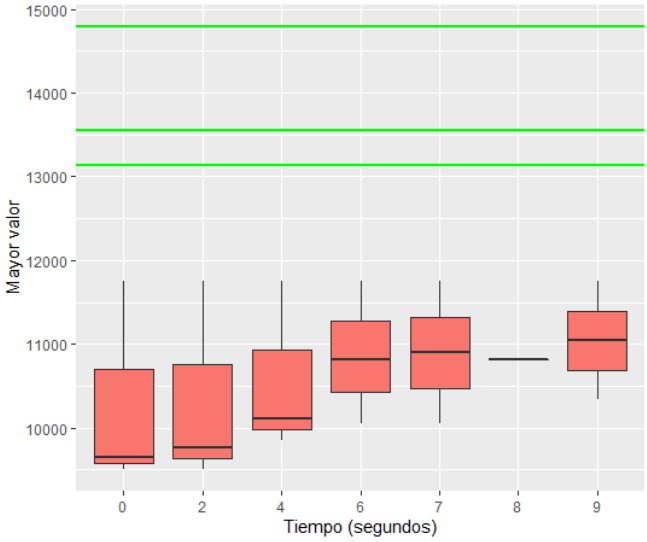
Cuadro 4: Resultados al aplicar la prueba `Wilcoxon`.

Valor de p	10	15
15	0,091	-
20	0,026	0,250

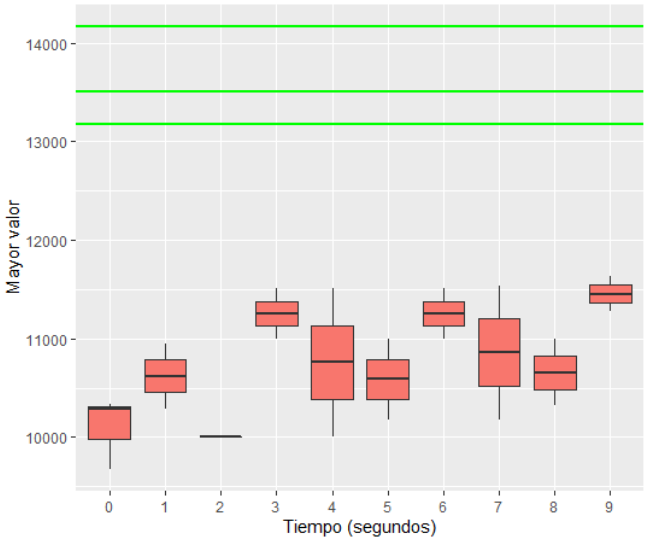
Hipótesis nula : Las medias son iguales en todos los grupos. Se rechaza esta hipótesis, si hay diferencias significativas de las medias.



(a) Población = 25



(b) Población = 35



(c) Población = 45

Figura 3: Mayor valor para cada tiempo de acuerdo a cada tamaño de población para el código 3 de la regla 1.

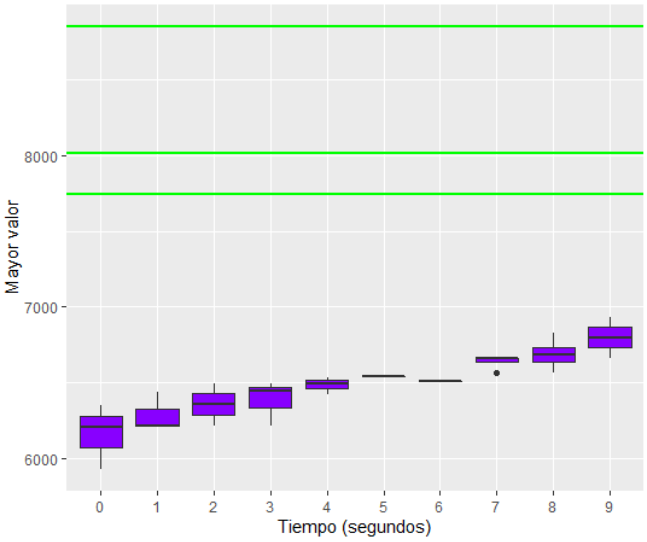
Cuadro 5: Resultados de la prueba estadística.

Outliers	0
Normalidad por grupo	25: $p = 0{,}0603$ / 35: $p = 0{,}0047$ / 45: $p = 0{,}0494$
Kruskal Wallis	$p = 0{,}2028$

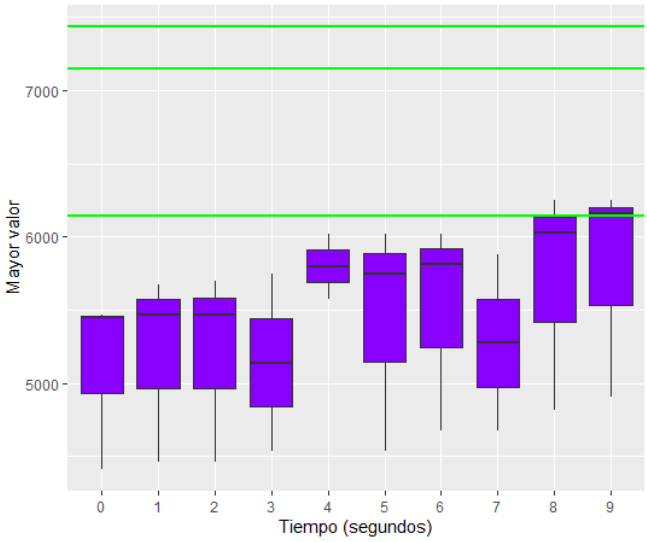
Cuadro 6: Resultados al aplicar la prueba `Wilcoxon`.

Valor de p	25	35
35	0,45	-
40	0,25	0,73

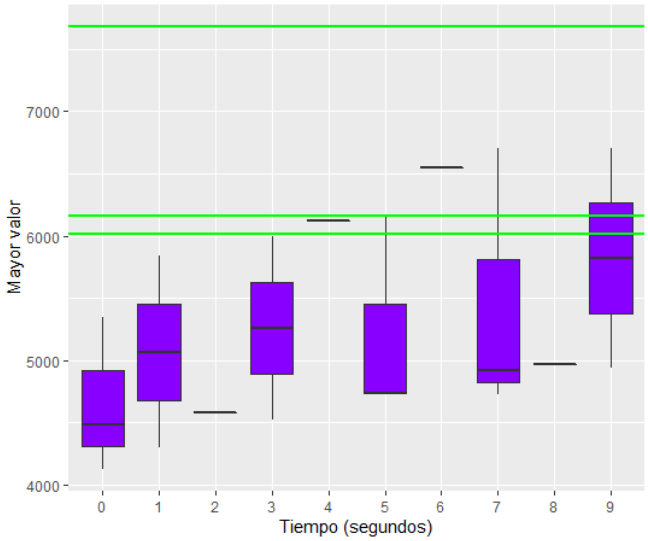
Hipótesis nula : Las medias son iguales en todos los grupos. Se acepta esta hipótesis.



(a) Probabilidad de mutación = 0.1



(b) Probabilidad de mutación = 0.3



(c) Probabilidad de mutación = 0.6

Figura 4: Mayor valor para cada tiempo de acuerdo a cada probabilidad de mutación para el código 1 de la regla 2.

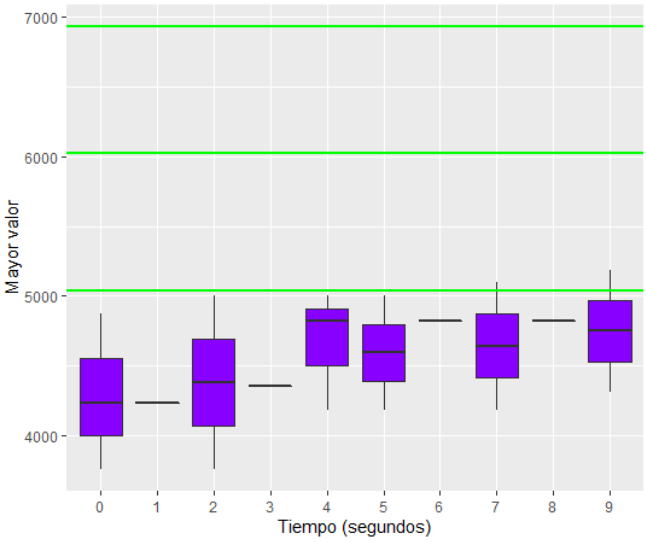
Cuadro 7: Resultados de la prueba estadística.

Outliers	1
Normalidad por grupo	0,1: $p = 0{,}295$ / 0,3: $p = 0{,}006$ / 0,6: $p = 0{,}039$
Kruskal Wallis	$p = 7{,}966 \times 10^{-9}$

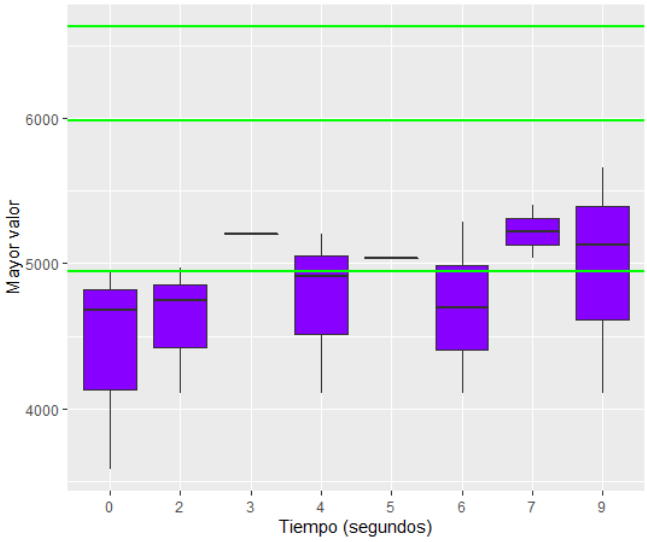
Cuadro 8: Resultados al aplicar la prueba `Wilcoxon`.

Valor de p	0,1	0,3
0,3	$5{,}7 \times 10^{-9}$	-
0,6	$6{,}6 \times 10^{-5}$	0,67

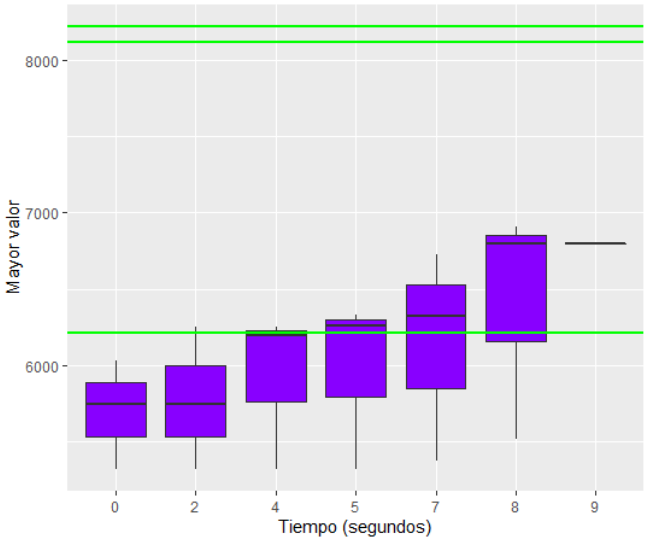
Hipótesis nula : Las medias son iguales en todos los grupos. Se rechaza esta hipótesis, si hay diferencias significativas de las medias.



(a) Cruzamientos = 10



(b) Cruzamientos = 15



(c) Cruzamientos = 20

Figura 5: Mayor valor para cada tiempo de acuerdo a cada cantidad de cruzamientos para el código 2 de la regla 2.

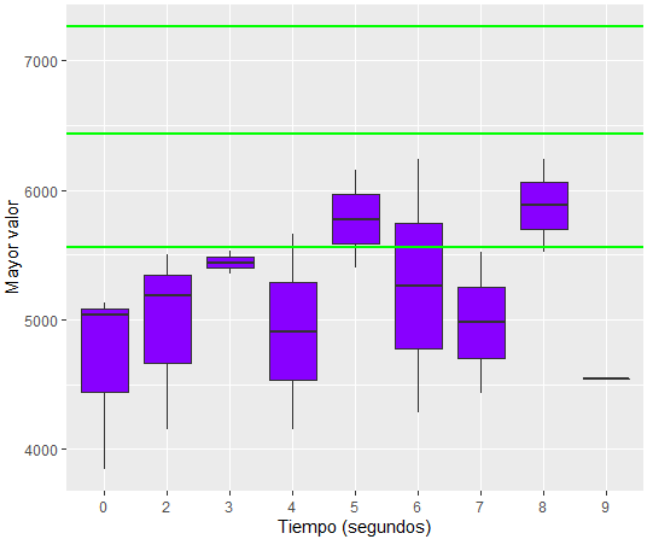
Cuadro 9: Resultados de la prueba estadística.

Outliers	0
Normalidad por grupo	10: $p = 0{,}054$ / 15: $p = 0{,}113$ / 20: $p = 0{,}042$
Kruskal Wallis	$p = 2{,}11 \times 10^{-8}$

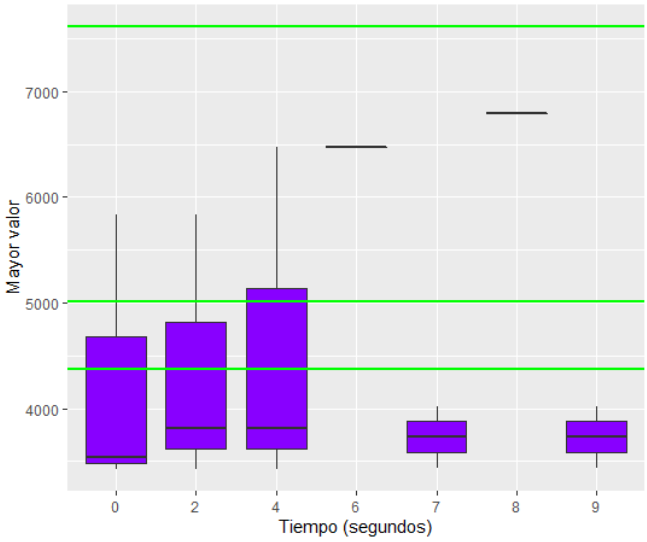
Cuadro 10: Resultados al aplicar la prueba `Wilcoxon`.

Valor de p	10	15
15	0,19	-
20	$6{,}4 \times 10^{-7}$	$2{,}4 \times 10^{-6}$

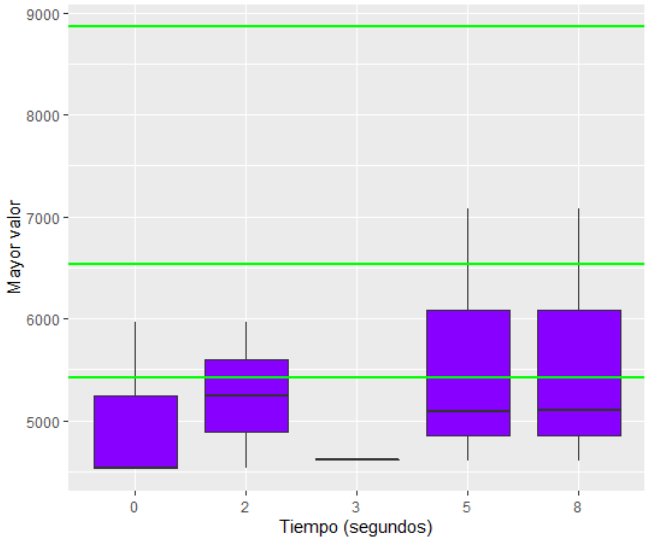
Hipótesis nula : Las medias son iguales en todos los grupos. Se rechaza esta hipótesis, si hay diferencias significativas de las medias.



(a) Población = 25



(b) Población = 35



(c) Población = 45

Figura 6: Mayor valor para cada tiempo de acuerdo a cada tamaño de población para el código 3 de la regla 2.

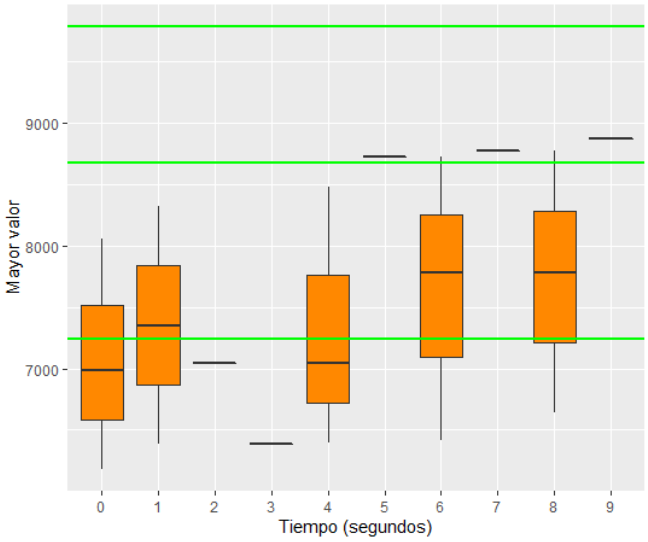
Cuadro 11: Resultados de la prueba estadística.

Outliers	0
Normalidad por grupo	25: $p = 0{,}210$ / 35: $p = 0{,}001$ / 45: $p = 0{,}005$
Kruskal Wallis	$p = 0{,}07854$

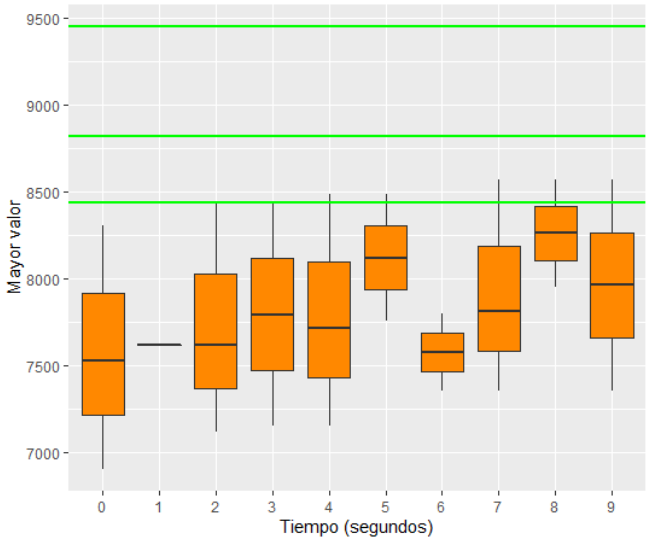
Cuadro 12: Resultados al aplicar la prueba `Wilcoxon`.

Valor de p	25	35
35	0,15	-
40	0,92	0,10

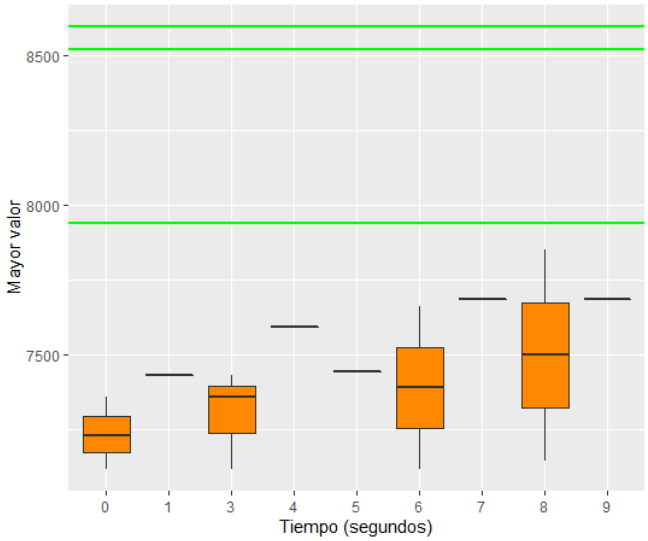
Hipótesis nula : Las medias son iguales en todos los grupos. Se acepta esta hipótesis.



(a) Probabilidad de mutación = 0.1



(b) Probabilidad de mutación = 0.3



(c) Probabilidad de mutación = 0.6

Figura 7: Mayor valor para cada tiempo de acuerdo a cada probabilidad de mutación para el código 1 de la regla 3.

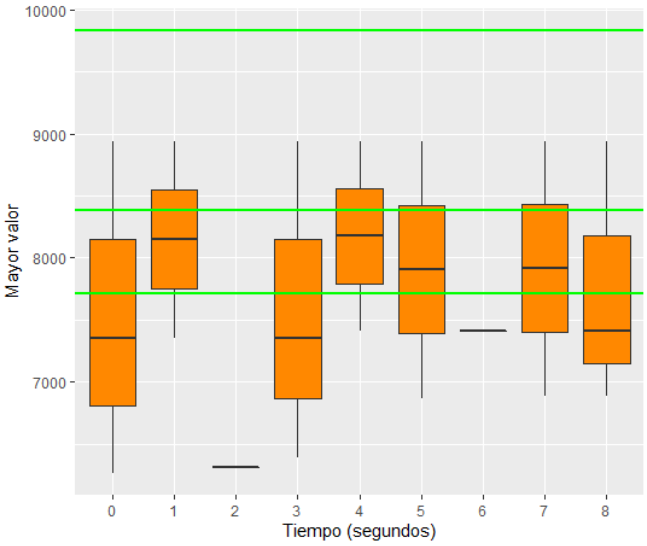
Cuadro 13: Resultados de la prueba estadística.

Outliers	0
Normalidad por grupo	0,1: $p = 0{,}0143$ / 0,3: $p = 0{,}0395$ / 0,6: $p = 0{,}2080$
Kruskal Wallis	$p = 0{,}2004$

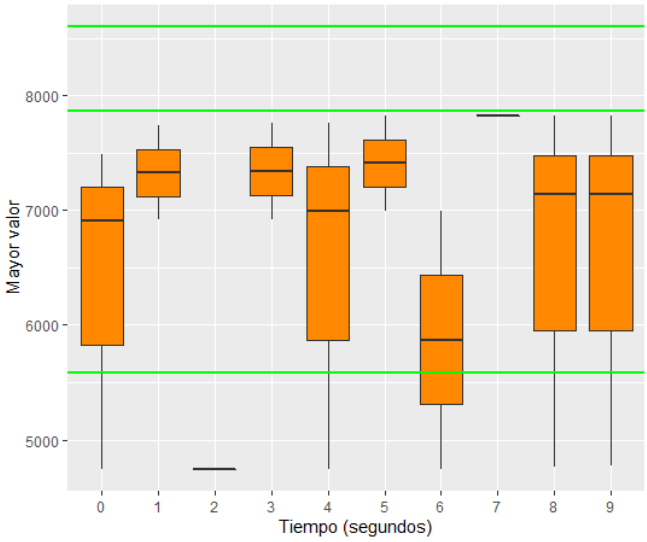
Cuadro 14: Resultados al aplicar la prueba `Wilcoxon`.

Valor de p	0,1	0,3
0,3	0,990	-
0,6	0,990	0,087

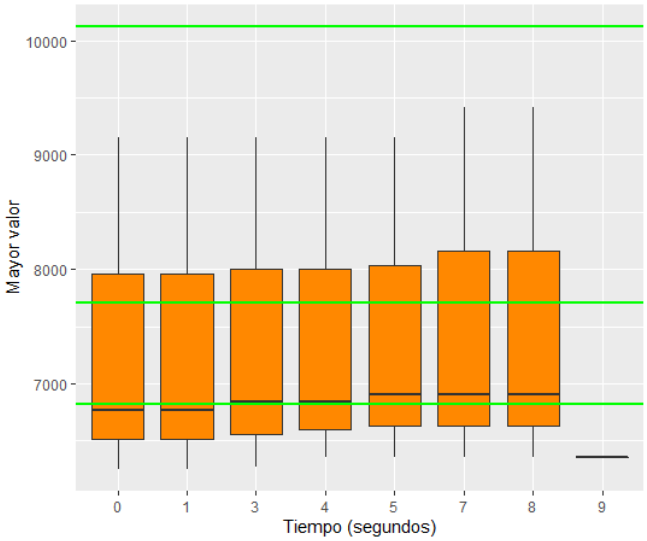
Hipótesis nula : Las medias son iguales en todos los grupos. Se acepta esta hipótesis.



(a) Cruzamientos = 10



(b) Cruzamientos = 15



(c) Cruzamientos = 20

Figura 8: Mayor valor para cada tiempo de acuerdo a cada cantidad de cruzamientos para el código 2 de la regla 3.

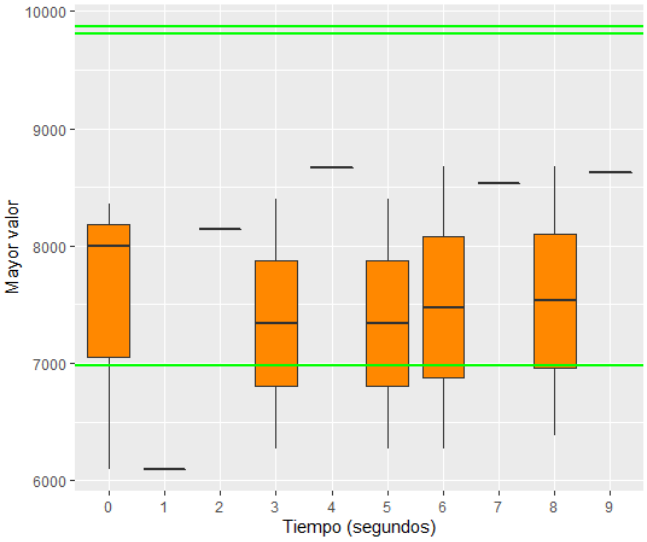
Cuadro 15: Resultados de la prueba estadística.

Outliers	0
Normalidad por grupo	10: $p = 00{,}00258$ / 15: $p = 0{,}00010$ / 20: $p = 0{,}00005$
Kruskal Wallis	$p = 0{,}3074$

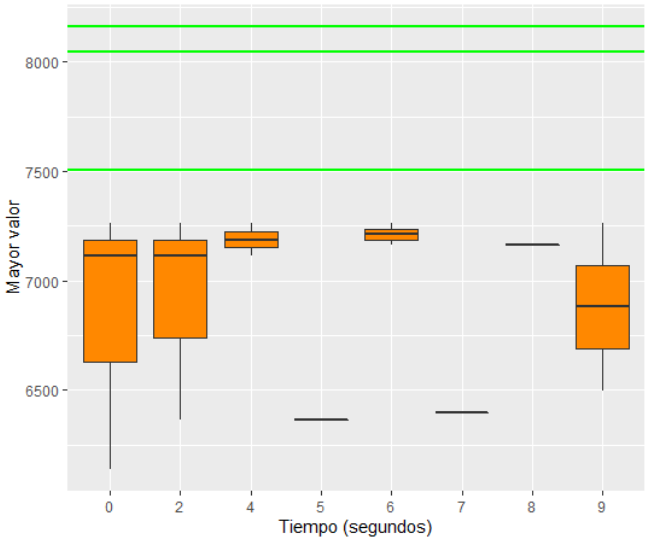
Cuadro 16: Resultados al aplicar la prueba `Wilcoxon`.

Valor de p	10	15
15	0,27	-
20	0,73	0,92

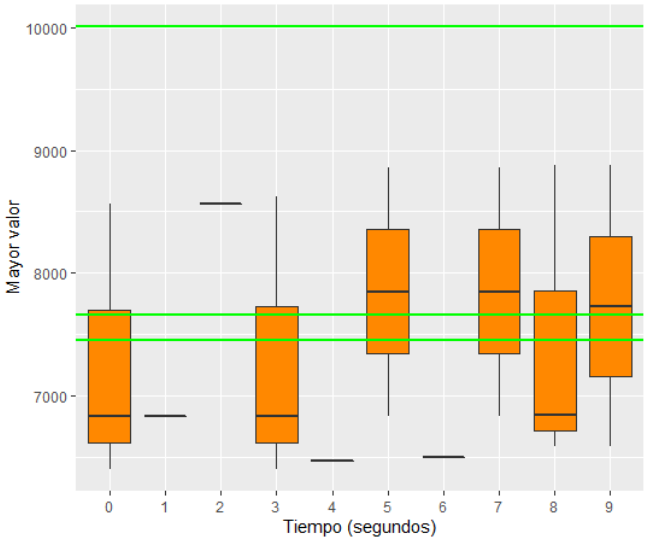
Hipótesis nula : Las medias son iguales en todos los grupos. Se acepta esta hipótesis.



(a) Población = 25



(b) Población = 35



(c) Población = 45

Figura 9: Mayor valor para cada tiempo de acuerdo a cada tamaño de población para el código 3 de la regla 3.

Cuadro 17: Resultados de la prueba estadística.

Outliers	0
Normalidad por grupo	25: $p = 0{,}00063$ / 35: $p = 0{,}00103$ / 45: $p = 0{,}00020$
Kruskal Wallis	$p = 0{,}5021$

Cuadro 18: Resultados al aplicar la prueba `Wilcoxon`.

Valor de p	25	35
35	0,48	-
40	1	1

Hipótesis nula : Las medias son iguales en todos los grupos. Se acepta esta hipótesis.

A continuación se muestra el código utilizado para realizar los diagramas caja-bigote y las prueba estadística `Kruskal Wallis` para el código 1 de la regla 1, este código también se aplica a las demás reglas únicamente cambiando los respectivos valores que se ocupan analizar:

```
library(ggplot2)
df$Segundo = as.factor(df$Segundo)
dfs = split.data.frame(df, f = df$Mutacion)

ggplot(dfs$`0.1`, aes(x= Segundo, y= Mejor)) +
  geom_boxplot(fill = "#F8766D")+
  labs(x = "Tiempo (segundos)", y = "Mayor valor", title = 'pm = 0.1')+
  geom_hline(aes(yintercept=Optimo), colour="green", size= 1)

ggplot(dfs$`0.3`, aes(x= Segundo, y= Mejor)) +
  geom_boxplot(fill = "#F8766D")+
  labs(x = "Tiempo (segundos)", y = "Mayor valor", title = 'pm = 0.3')+
  geom_hline(aes(yintercept=Optimo), colour="green", size= 1)

ggplot(dfs$`0.6`, aes(x= Segundo, y= Mejor)) +
  geom_boxplot(fill = "#F8766D")+
  labs(x = "Tiempo (segundos)", y = "Mayor valor", title = 'pm = 0.6')+
  geom_hline(aes(yintercept=Optimo), colour="green", size= 1)

library(tidyverse)
library(ggpubr)
library(car)
library(rstatix)
library(rapportools)
library(readr)
library(gridExtra)

#PRUEBA ESTADISTICA...
#Estadisticas descriptivas
df %>%
  group_by(Mutacion) %>%
  get_summary_stats(Mejor, type = "mean_sd")

#SUPUESTOS PARA ANOVA
#1:Outliers
df %>%
  group_by(Mutacion) %>%
  identify_outliers(Mejor)

#2:Normalidad por Shapiro
df %>%
  group_by(Mutacion) %>%
  shapiro_test(Mejor)

#3:Homogeneidad de varianza con prueba Levene
df %>%
  levene_test(Mejor~Mutacion)

#PRUEBA ESTADISTICA KRUSKAL WALLIS
kruskal.test(Mejor ~ Mutacion, data = df)

#PRUEBA WILCOXON
pairwise.wilcox.test(df$Mejor, df$Mutacion)
```

Listing 7: Código para las pruebas estadísticas `Kruskal Wallis` y `Wilcoxon`.

3. Conclusión

Con base en los diagramas caja-bigote obtenidos puedo concluir que en la regla 1 conforme pasan los segundos se logra alcanzar el valor óptimo en probabilidades de mutación más grandes, lo mismo sucede con la variación de los cruzamientos, también se logra alcanzar el valor óptimo conforme pasan los segundos y con respecto a la variación del tamaño de población éste no tiene tanta influencia en la posibilidad de alcanzar el valor óptimo. Para la regla 2 de igual manera conforme avanzan los 10 segundos se puede llegar al valor óptimo sobre todo con probabilidades de mutación grandes y en cuando a los cruzamientos también se observa este comportamiento, solamente en el tamaño de población no se tiene una relación entre el tiempo y los valores mayores obtenidos. En la regla 3 se puede observar un comportamiento diferente, en la mayoría de las combinaciones desde el segundo 1 se logra alcanzar el valor óptimo y este comportamiento se mantiene.

En general ésta práctica se me dificultó la parte estadística pero con el apoyo de mis compañeros finalmente pude realizarla.

Referencias

[1] Eduardo Navarro. tarea10pruebas, 2021. URL https://github.com/eduartilon/Simulacion/blob/main/tarea10/tarea10pruebas/Contadorsegundos.R.

[2] Elisa Schaeffer. Genetic algorithm, 2021. URL https://github.com/satuelisa/Simulation/blob/master/GeneticAlgorithm/perfGA.R.