Práctica 10: algoritmo genético

En la práctica 10 llamada algoritmo genético simula el fenómeno de evolución para una población `p` de `init` cantidad de miembros, utilizando dos métodos: la mutación y la reproducción. La mutación (Fig.1) crea un nuevo miembro cuando un miembro de la población se duplica y cambia alguna de sus características originales, este fenómeno se presenta cuando la probabilidad de mutar de un individuo (dictada al azar) es menor a la probabilidad de mutación `pm;` mientras que en la reproducción (Fig. 2) crea dos nuevos miembros por la combinación de características de sus padres.

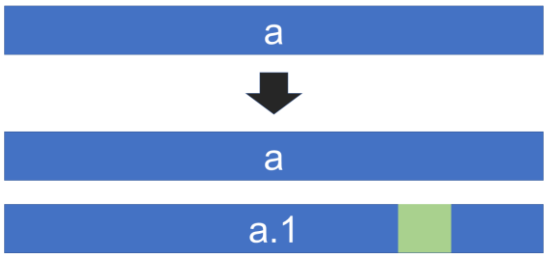


Figura 1 Representación visual del fenómeno de mutación

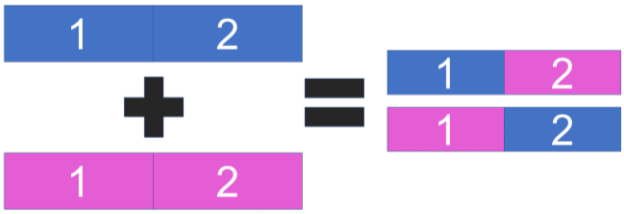


Figura 2 Diagrama del método de reproducción

Este tipo de proyectos pueden convertirse en proyectos complejos debido a la gran cantidad de posibles soluciones, es por ello que se ha implementado la optimización `knapsack` para obtener el valor objetivo del proyecto.

Objetivo

Identificar las partes del código que tienen potencial para ser paralelizadas y paralelizarlas usando los paquetes `Parallel` o `doParallel` con el objetivo de mejorar el tiempo de ejecución de la simulación.

Desarrollo del código

Debido a la extensión del código paralizado, gran parte del mismo será omitido y sólo se discutirán las modificaciones realizadas; sin embargo, el código completo puede encontrarse aquí. Con el objetivo de paralelizar esta simulación se creación tres funciones: `para.mut`, `para.rep` y `para.obj`, las cuales substituyen a las instrucciones `for (i)` de las líneas de código original. La paralelización se llevó a cabo utilizando el paquete `doParallel`.

La función `para.mut` tiene como objetivo la creación de los nuevos individuos que han mutado, puede observarse que la creación de estos individuos es dictada por una condicional `runif(1) < pm`, la cual se discutió anteriormente. La instrucción `unlist` es útil para transformar una lista en un vector para su posterior unión con los individuos originales.

```
> para.mut<-function(i){
+   if (runif(1) < pm) {
+     return(unlist(mutacion(p[i,],n)))
+   }
+ }
```

```
+   p<-rbind(p,foreach(i=1:tam,.combine = rbind)%dopar% para.mut(i))
```

La función `para.rep` tiene como objetivo simular el fenómeno de reproducción. Se espera que durante esta simulación se divida la población `p` original en dos grupos, los cuales fungirán como donadores de características, es decir, ellos heredarán una mitad de sus características individuales a su primer hijo, y la segunda a su segundo hijo. Aumentando así el tamaño de la población a más de 300 individuos.

```
> para.rep<-function(i){
+   padres <- sample(1:tam, 2,replace=TRUE)
+   hijos <- reproduccion(p[padres[1],], p[padres[2],], n)
+   p.hijo<- hijos[1:n] # primer hijo
+   s.hijo<- hijos[(n+1):(2*n)] # segundo hijo
+   son<-rbind(p.hijo,s.hijo)
+   return(son)
+ }
```

```
+   p<-rbind(p,foreach(i=1:rep,combine=rbind)%dopar% para.rep(i))
```

La función `para.obj` permitirá conocer cuales combinaciones tienen una mejor solución, para su posterior elección.

```
> para.obj<-function(i){
+   obj <- objetivo(p[i,], valores)
+   fact <-factible(p[i,], pesos, capacidad)
+   datos<-cbind(obj,fact)
```

```
+   return(datos)
+ }
```

```
+   p<- data.frame(sapply(p, function(x) as.numeric(as.character(x))))
+   p<-cbind(p,foreach(i=1:tam,.combine=rbind)%dopar% para.obj(i))
```

Para medir el tiempo de ejecución del programa original y el programa paralelizado se hizo uso de la instrucción `source`, la cual permite llamar al código seleccionado y ejecutarlo, para sí obtener el tiempo de ejecución del mismo. Estos datos se almacenaron dentro del cuadro de datos `resultados`. Debido a las características del procesador, se decidió conocer el tiempo de ejecución para una sola generación de individuos, asumiendo que el tiempo para otras generaciones seria muy cercano a este ya que se analiza la misma cantidad de información.

```
> for (replicas in 1:ciclos){
+   for (init in seq(200,500,100)){
+
+     source('~/GitHub/Simulacion/Simulacion/P10/Cod_P10_T10.R')
+     t.paralel<-cbind("paralelo",replicas, init,t)
+
+     source('~/GitHub/Simulacion/Simulacion/P10/Cod_P10.R')
+     t.original<-cbind("original",replicas,init,t)
+
+     resultados<-rbind(resultados,t.paralel,t.original)
+   }
+ }
> save.image(file="Resultados_Tarea.RData")
```

Para una mejor visualización en la diferencia de tiempos de ejecución se procedió a graficar tales valores en un gráfico tipo caja-bigote (Fig. 3) utilizando el paquete `gggplot2`.

```
> library(ggplot2)
> colnames(resultados)<-c("Tipo","Replica","Init","Tiempo")
>                                                    resultados$Tiempo<-
as.numeric(levels(resultados$Tiempo))[resultados$Tiempo]
> resultados$Tipo <- as.factor(resultados$Tipo)
> resultados$Init <- as.factor(resultados$Init)
>
> png("Variacion_pob.png")
> ggplot(data=resultados, aes(x = Init, y= Tiempo,fill = Tipo)) +
+   geom_boxplot(position=position_dodge(1))+
+   ylab("Tiempo (s)") +
+   xlab("Población")
> dev.off()
```

En la figura 3 se observa una clara diferencia entre los tiempos de ejecución del programa paralelizado y el original. Esta diferencia aumenta conforme el número de

individuos que integran la población inicial aumenta. Por este motivo se recomienda el uso de el código paralelizado incluso en valores “pequeños” de individuos.

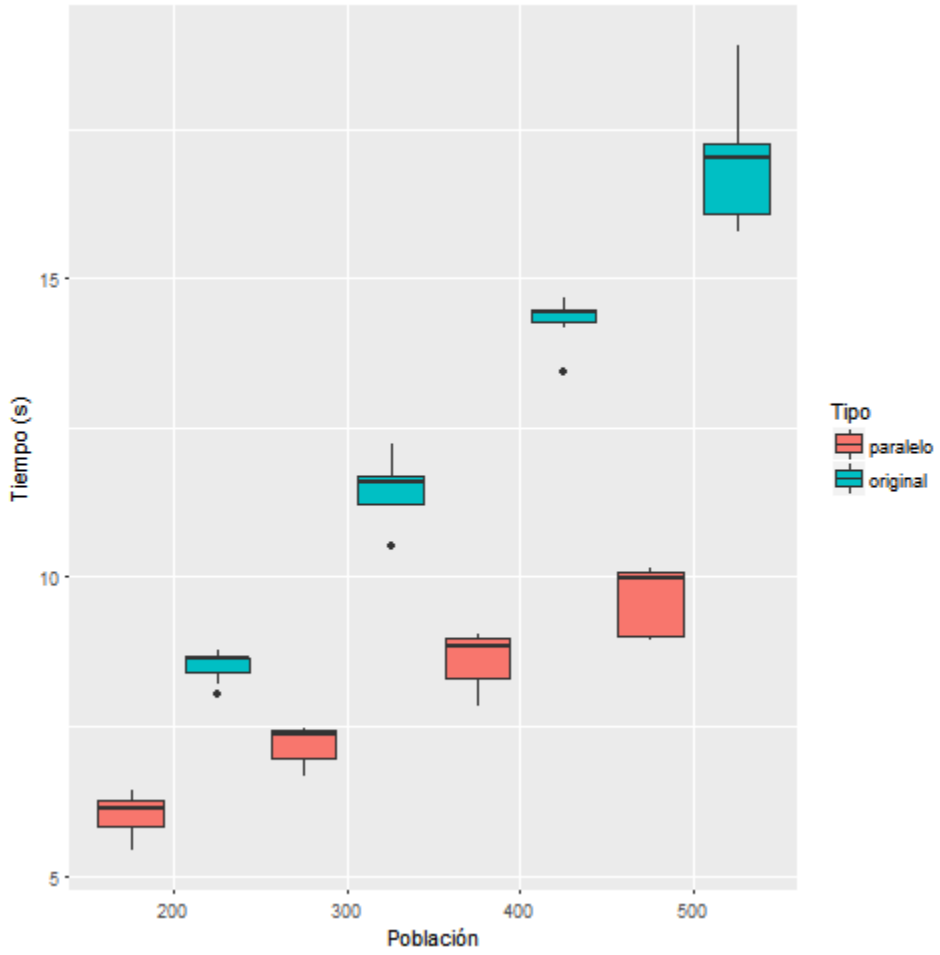


Figura 3 Representación gráfica de un diagrama caja-bigote. Se presentan los valores de tiempo de ejecución durante el experimento.

Reto 1

El reto 1 consiste en que el fenómeno de reproducción se lleve a cabo en función a una probabilidad de reproducción que se encuentra en función al valor objetivo que pueden alcanzar los individuos con las características n que poseen. Es decir, individuos con mayor valor de la función objetivo serán más propensos a reproducirse que aquellos con una un valor menor en su función objetivo. Este tipo de selección se le conoce como selección ruleta.

Objetivo

Observar los cambios en el valor de la función objetivo al agregar una selección ruleta dentro de la función de reproducción.

Desarrollo del código

Después de realizar el proceso de mutación, se procedió a la evaluación de la función objetivo. Resultando en un vector `elites` de 200 elementos, a los cuales se le realizaron algunas operaciones aritméticas para asegurar que el valor se los mismos se encuentre entre 0 y 1, con el objetivo de usarlo como una probabilidad con la cual el individuo puede fungir como padre.

```
+   elites<-foreach(i=1:tam,.combine = c)%dopar% objetivo(p[i,], valores)
+   elites <- elites / sum(elites)
```

Para la realización de este código se realizaron algunas modificaciones a la función `para.rep` en la cual al momento de declarar `padres` interviene una `prob = elites` previamente calculada, la cual permite la reutilización de un mismo elemento, procurando que elementos con más alta probabilidad se reproduzcan más de una vez.

```
> para.rep.m<-function(i){
+   padres <- sample(1:tam, 2, prob=elites,replace=TRUE)
+   hijos <- reproduccion(p[padres[1],], p[padres[2],], n)
+   p.hijo<- hijos[1:n] # primer hijo
+   s.hijo<- hijos[(n+1):(2*n)] # segundo hijo
+   son<-rbind(p.hijo,s.hijo)
+   return(son)
+ }
```

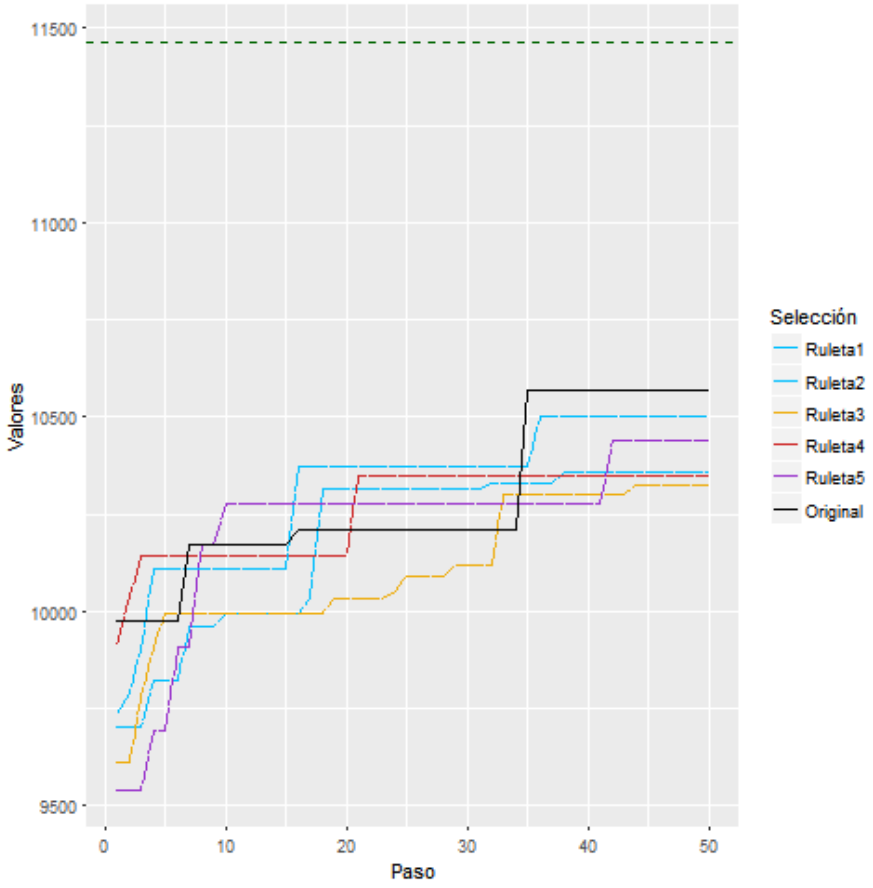
Se ejecutó el código de la tarea base usando la instrucción `source` con el objetivo de observar una diferencia entre ambos métodos de reproducción.

```
> source('~/GitHub/Simulacion/Simulacion/P10/Cod_P10_T10.R')
> tarea<-cbind(mejores,c(1:tmax))
> tarea<-as.data.frame(tarea)
> colnames(tarea)<-c("Valores","Paso")
> colnames(resultados)<-c("Replica","Valores","Paso")
> resultados$Replica<-as.factor(resultados$Replica)
```

Se prosiguió a crear un grafico para observar alguna diferencia entre ambos métodos.

```
> png("Seleccion_ruleta.png")
> ggplot()+
+
geom_line(data=resultados,aes(x=resultados$Paso,y=resultados$Valores,colo
r=resultados$Replica),size=0.6,linetype="F1")+
+          geom_hline(yintercept=optimo,    linetype="dashed",    color    =
"darkgreen",size=0.6)+
+
geom_line(data=tarea,aes(x=tarea$Paso,y=tarea$Valores,color="black"),size
=0.6)+
+   xlab("Paso")+    ylab("Valores")+
+             scale_color_manual(name     =     "Selección",     values     =
c("deepskyblue","deepskyblue1",
+           "darkgoldenrod2","firebrick3","darkorchid","black"),
+                                                  labels     =c("Ruleta1",
"Ruleta2","Ruleta3","Ruleta4","Ruleta5","Original")) +
+     guides(color = guide_legend(order = 1))
>     dev.off()
```

Visualmente en la figura 4 no es posible encontrar una diferencia significativa entre ambos métodos. Esto se debe posiblemente a la gran cantidad de datos que “pelean” por una oportunidad de reproducción.



Referencias

[1] Elisa.dyndns-web.com. (2017). P10 — R paralelo — Schaeffer. [online] Available at: http://elisa.dyndns-web.com/teaching/comp/par/p10.html [Accessed 17 Oct. 2017].