Práctica 10: Algoritmo genético

Alumno: José Adrian Garcia Fuentes
Profesor: Satu Elisa Schaeffer

Universidad Autónoma de Nuevo León. Facultad de Ingeniería Mecánica y Eléctrica.
30/abril/2021

Resumen

En esta práctica se modifica un algoritmo genético con la finalidad de resolver el problema de la mochila y se compara la mejor solución alcanzada con la solución optima obtenida con un método exacto.

Palabras Claves: *Algoritmo, Knapsack.*

1. Introducción

En la práctica 10 se utiliza el problema de la mochila este es un problema clásico de optimización particularmente de programación entera donde la tarea consiste en seleccionar un subconjunto de objetos de tal forma que no exceden la capacidad de la mochila en términos de la suma de los pesos de los objetos incluidos y que el valor total de los objetos incluidos sea lo máximo posible [1], mediante el problema de la mochila se implementó un algoritmo genético el cual representa posibles soluciones que satisfacen a un problema, creando valores óptimos del problema mediante una buena codificación a las diferentes variables que podrían cuantificarse para dicho método para fines de esta práctica se encontrara representado como un vector de verdadero y falso, indicando cuales objetos vamos a incluir en la mochila ($True$ o 1 llevamos el objeto, $False$ o 0 se descarta el objeto).

2. Objetivo

- Cambiar selección de mutación y de padres para reproducción a que use selección de ruleta [1].
- Genere instancias con tres distintas reglas [1].
- Determinar para cada uno de los tres casos a partir de qué tamaño de instancia el algoritmo genético es competitivo con el algoritmo exacto [1].

3. Metodologia

La metodología empleada se realizó a través de Rstudio [2] llevando a cabo los pasos señalados en la *Práctica 10: Algoritmo genético* [1], a partir del código en el repositorio de Schaeffer [3], se realizaron modificaciones. El código completo de la metodología empleada se encuentra en el repositorio de GitHub [4].

4. Resultados

Simulando la aplicación del problema de la mochila, conocido en ingles como *Knapsack problem*, donde se tienen un contenedor de capacidad C limitada y se tienen N elementos que se pueden meter en el contenedor , cada elemento tiene un valor de beneficio b_i y un valor de capacidad c_i que ocupan en el contenedor, se busca tener en el contenedor aquellos que nos maximizan el beneficio máx , donde x_i es la decisión de tener o no el elemento en el contenedor sin exceder la capacidad C.

A continuación se muestra la función de generador de valores que fue modificada.

```
generador.valores <- function(cuantos, min,
    max) {
  return(sort(round(normalizar(rnorm(cuantos))
    * (max - min) + min)))
}
```

Se realizaron cambios para la creación de variables.

```
p1 <- poblacion.inicial(n, init)
p<- p1
tam <- dim(p)[1]
assert(tam == init)
pm <- sum(runif(tam) < 0.05)
rep <- 50
tmax <- 50
mejorescon <- double()
```

Se crean los vectores de factibilidad y objetivos.

```
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))
  }
```

Se generan las mutaciones.

```
  for (i in 1:tam) {
    prob.mut[i] = 1/(obj[i]*(fact[i]+1)*sum(
    obj*(fact+1)))
  }
  mutantes<-sample(1:tam, pm, prob = prob.mut)
  for (i in mutantes) {
    p <- rbind(p, mutacion(p[i,], n))
  }
```

Se llevan acabo las reproducciones.

```
for (i in 1:tam) {
    prob.reprod[i] = obj[i]*(fact[i]+1)/sum(
    obj*(fact+1))
  }

```

Creación de variables y fabricación de las interacciones del algoritmo genético sin ruleta, aplicando una probabilidad de mutación, se determina una cantidad fija de reproducciones.

```
mejorescon <- c(mejorescon, mejor)
}
p<- p1
mejoressin <- double()
for (iter in 1:tmax) {
  p$obj <- NULL
  p$fact <- NULL
  for (i in 1:tam) {
    if (runif(1) < pm) {
      p <- rbind(p, mutacion(p[i,], n))
    }
  }
  for (i in 1:rep) {
    padres <- sample(1:tam, 2, replace=FALSE)
    hijos <- reproduccion(p[padres[1],], p[
    padres[2],], n)
    p <- rbind(p, hijos[1:n])
    p <- rbind(p, hijos[(n+1):(2*n)])
  }
  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)
  mejoressin <- c(mejoressin, mejor)
}
```

Los resultados obtenidos al correr el código se muestran en la figura 1 se analizan los valores óptimos así como el mayor valor alcanzado en la tabla 1.

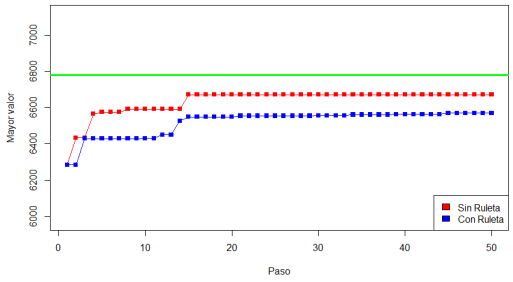

Figura 1: Gráfico competitivo con ruleta y sin ruleta de la primera instancia.

Tabla 1: Mayor valor alcanzado vs valor optimo

Mejor	Optimo	Porcentaje
6672	6780	0,01

Ahora en base al primer código modificado se realizaron ligeras adecuaciones con el fin de cumplir con los objetivos de la práctica y se generen distintas reglas, los cambios realizados se muestran a continuación y se muestra en la figura 2, se analizan los valores óptimos así como el mayor valor alcanzado en la tabla 2, se observa una mejora en los valores óptimos aplicando la ruleta y sin ruleta en comparación con la capacidad máxima.

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

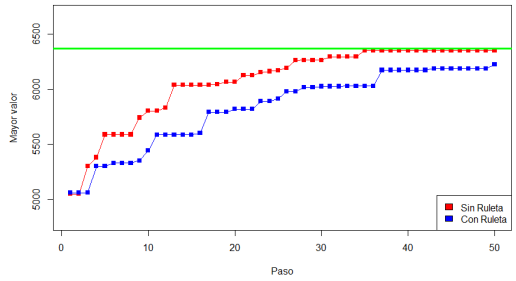

Figura 2: Gráfico competitivo con ruleta y sin ruleta de la segunda instancia.

Tabla 2: Mayor valor alcanzado vs valor optimo

Mejor	Optimo	Porcentaje
6352	6367	0,002

Con el fin de cumplir con la tercera y ultima instancia se modifica la función generador de valores observándose valores menos favorables debido aque el porcentaje es menor en comparación con los valores anteriores, los datos obtenidos se muestran en la tabla 3 y se representan en la figura 3.

```
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)
}
```

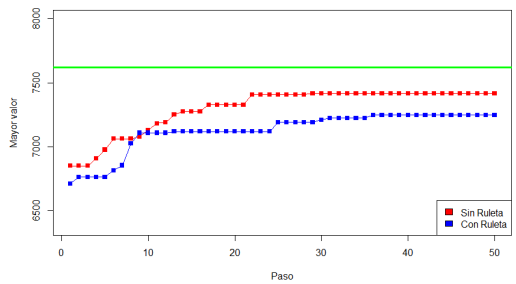

Figura 3: Gráfico competitivo con ruleta y sin ruleta de la tercera instancia.

Tabla 3: Mayor valor alcanzado vs valor optimo

Mejor	Optimo	Porcentaje
7417,10	7620,28	0,02

5. Reto 1

Extender la selección de ruleta a la fase de supervivencia: en vez de quedarnos con las mejores soluciones, cada solución tiene una probabilidad de entrar a la siguiente generación que es proporcional a su valor de la función objetivo, incorporando el sí o no es factible la solución en cuestión, permitiendo que los k mejores entre las factibles entren siempre (donde k es un parámetro). Estudia nuevamente el efecto de este cambio en la calidad de la solución en los tres casos.

```
prob.supervivencia <- NULL


for (i in 1:tam) {
    prob.supervivencia[i] = obj[i]*(fact[i]+1)
    /sum(obj*(fact+1))
  }
  supervivientes<-sample(1:tam,init,prob =
    prob.supervivencia)
  p <- p[supervivientes,]

```

El código fue modificado para cada una de las 3 instancias anteriores realizadas en la tarea base las figuras 4, 5, 6 muestran los valores obtenidos respectivamente, comparándolos con la tarea base observamos que los resultados son muy variables sin embargo la función de generador de pesos nos arroja un valor ideal para los casos con ruleta y sin ruleta, los valores se representan en la tabla 4.

Tabla 4: Mayor valor alcanzado vs valor optimo

	Mejor	Optimo	Porcentaje
fig.4	8982	9417	0,04
fig.5	4318	4939	0,12
fig.6	6476	7258	0,1

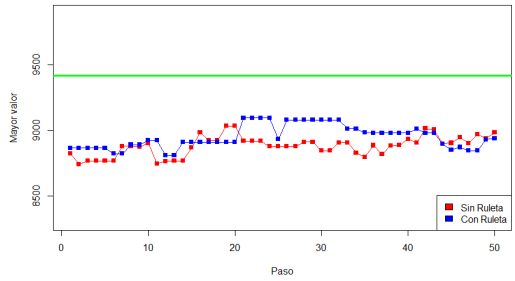

Figura 4: Gráfico competitivo con probabilidad de entrar ala siguiente generación.

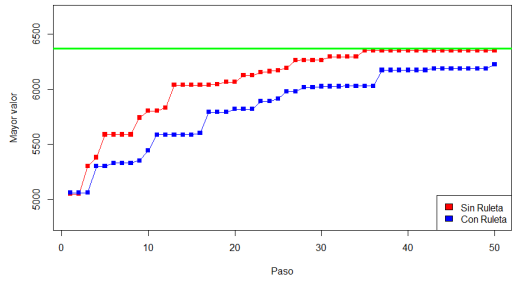

Figura 5: Gráfico competitivo con probabilidad de entrar ala siguiente generación.

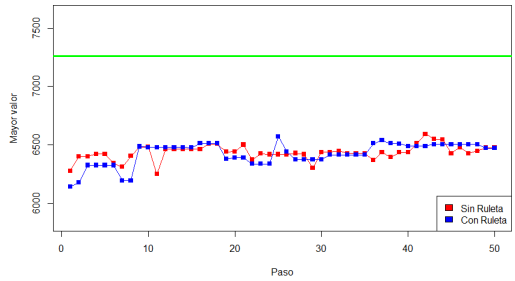

Figura 6: Gráfico competitivo con probabilidad de entrar ala siguiente generación.

6. Reto 2

Paraleliza el algoritmo genético y estudia los efectos en su tiempo de ejecución con pruebas estadísticas y visualizaciones, variando el número de objetos en la instancia.

```
cluster <- makeCluster(detectCores() - 1)
clusterExport(cluster,"factible")
clusterExport(cluster,"objetivo")
clusterExport(cluster,"normalizar")
clusterExport(cluster,"poblacion.inicial")
clusterExport(cluster,"mutacion")
clusterExport(cluster,"reproduccion")
datos=data.frame()
for(init in c(50,100,200))

```

```
clusterExport(cluster,"n")
  clusterExport(cluster,"capacidad")
  clusterExport(cluster,"init")
  clusterExport(cluster,"pesos")
  clusterExport(cluster,"valores")

```

A partir de la página RDocumentation [5] se encontraron algunas funciones con el fin de paralelizar el código de Scaeffer [6], sin embargo al correr el código se encuentra un error en la interpretación de la imagen.

```
      clusterExport(cluster,"p")
      mutan=sample(1:tam,round(pm*tam))
      p <- rbind(p,(t(parSapply(cluster,
mutan,function(i){return(mutacion(unlist(p
[i,]), n))}))))
      clusterExport(cluster,"tam")
      clusterExport(cluster,"p")
      padres <- parSapply(cluster,1:rep,
function(x){return(sample(1:tam, 2,
replace=FALSE))})
      clusterExport(cluster,"padres")
      hijos <- parSapply(cluster,1:rep,
function(i){return(as.matrix(unlist(
reproduccion(p[padres[1,i],], p[padres[2,i
],], n)),ncol=n))})
      p = rbind(p,hijos)
      tam <- dim(p)[1]
      clusterExport(cluster,"p")
        obj=parSapply(cluster,1:tam,function
(i){return(objetivo(unlist(p[i,]), valores
))})
       fact=parSapply(cluster,1:tam,
function(i){return(factible(unlist(p[i,]),
 pesos, capacidad))})

```

7. Conclusión

el algoritmo genético se puede mejorar al paralelizarlo, esto se nota en los tiempos de ejecución donde el tiempo de ejecución del codigo paralelizado será menor que el codigo secuencial.

Referencias

[1] E. Schaeffer, "Práctica 10: algoritmo genético," mayo 2021. https://https://elisa.dyndns-web.com/teaching/comp/par/p10.html.

[2] J. J. Allaire, "Rstudio," mayo 2021. https://rstudio.com.

[3] E. Schaeffer, "Práctica 10: algoritmo genético," mayo 2021. https://github.com/satuelisa/Simulation/tree/master/GeneticAlgorithm.

[4] J. A. Garcia mayo 2021. https://github.com/fuentesadrian/SIMULACION-DE-NANOMATERIALES/tree/main/Tarea%2010.

[5] Datacamp, "clusterapply aplicar operaciones mediante clústeres," mayo 2021. https://www.rdocumentation.org/packages/parallel/versions/3.6.2/topics/clusterApply.

[6] E. Schaeffer, "Práctica 10: algoritmo genético," mayo 2021. https://github.com/fuentesadrian/Simulation/tree/master/MultiAgent.