Universidad Galileo
Msc. Investigación de Operaciones
Simulación y Modelación I
Ing. Carlos Zelada

PROYECTO FINAL:
COMPAÑÍA DE CARGA AÉREA

Jully Jesmin Berganza López
21000183

Guatemala, 18 de septiembre de 2022.

INTRODUCCIÓN

Al definir logaritmo se entiende que es una serie de pasos las cuales están organizadas que se debe seguir, para la resolución de problema. Ahora, la definición de un algoritmo genético está basado en lo que es la vida real, en la reproducción e imitación para la solución y optimización de un problema.

Los algoritmos genéticos son parte de la rama de la inteligencia artificial; ya que estos están diseñados para resolver un problema específico, para lo cual fue diseñado, utilizando programas. Estos programas también están diseñados para imitar lo que la inteligencia natural hace, para poder optimizar el tiempo y el esfuerzo de la natural.

Los logaritmos genéticos, su principal objetivo es el poder optimizar la búsqueda de miles de datos, buscando la mejor solución; dependiendo de la necesidad del uso.

DESCRIPCIÓN DEL PROBLEMA

Una compañía de carga aérea tiene un avión que transporta carga desde Guatemala a Estados Unidos diariamente. Antes del vuelo recibe propuestas de pago para el envío de diferentes clientes. Los clientes envían el peso y cuánto están dispuestos a pagar.

La aerolínea está restricta por la capacidad de carga del avión. La compañía aérea tiene que seleccionar un subconjunto de paquetes que su peso total sea menor a la capacidad de carga del avión con el objetivo de maximizar su ganancia.

Nuestro avión tiene como capacidad 500 unidades de peso. A continuación, se presentan la lista de paquetes del día de hoy:

ID Paquete	peso	pago
1	55	70
2	31	32
3	43	36
4	23	65
5	48	49
6	40	62
7	53	25

8	54	47
9	39	78
10	77	65
11	39	23
12	59	45
13	53	43
14	28	44
15	71	71
16	36	67
17	62	48
18	30	36
19	48	76
20	42	2

Su tarea como consultor es dar a la compañía aérea el listado de paquetes que tiene que transportar el avión, así asegurando el rendimiento óptimo.

RESOLUCIÓN DEL PROBLEMA

Logaritmos Genéticos

1. Generar nueva población

Para generar una nueva población se tomó la tabla mencionada en el problema. Esta población debe tener un conjunto de variables, que, para los logaritmos genéticos, se le denominan Genes. Estos genes o variables se unen entre sí para generar un cromosoma, este cromosoma será el resultado del problema.

	peso	pago
1	55	70
2	31	32
3	43	36
4	23	65
5	48	49
6	40	62
7	53	25
8	54	47
9	39	78
10	77	65

	peso	pago
11	39	23
12	59	45
13	53	43
14	28	44
15	71	71
16	36	67
17	62	48
18	30	36
19	48	76
20	42	2

2. Se evaluó el fitness

En este paso se crea una función que determina si la persona es buena o no. Evalúa las variables si serán seleccionadas según su puntaje de aptitud.

En este problema se calcula el costo total de la población.

```
#2. Cálculo de Fitness (cálculo del costo total)
population <- t(sapply(1:500,function(x,w){chromosome(w)},w=weights_bids[,2] ))
population <- cbind(population,population %*% weights_bids[,1])
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21
son	1	1	0	0	1	0	1	1	0	0	0	1	1	0	0	1	1	0	1	1	54
son.1	0	1	0	0	0	1	1	0	1	0	0	0	0	0	0	0	1	0	1	1	31
son.2	1	1	1	0	0	1	1	1	0	1	0	0	0	0	1	0	0	0	0	1	46
son.3	1	0	0	0	0	0	0	1	0	1	1	1	0	1	0	1	0	0	0	0	34
son.4	0	0	1	1	0	0	0	1	0	0	1	0	0	0	0	1	1	1	1	0	33
son.5	0	0	0	0	0	1	1	1	0	1	0	0	0	0	0	1	1	0	1	1	41
son.6	0	1	1	1	0	0	0	0	1	1	1	0	1	0	0	1	1	0	0	1	44
son.7	0	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	0	1	1	0	41
son.8	1	0	1	1	0	0	0	0	1	0	1	1	0	0	0	1	0	0	0	0	29
son.9	1	1	1	0	0	1	1	0	0	0	1	1	1	0	0	1	1	0	0	1	51
son.10	0	1	0	1	1	1	0	1	0	1	0	1	0	0	0	1	1	0	1	0	47
son.11	0	1	0	0	1	0	1	1	0	0	1	1	1	0	0	1	0	0	0	0	37
son.12	1	1	0	0	0	1	0	1	0	1	1	1	1	1	0	0	0	0	0	1	47
son.13	0	0	1	1	1	0	0	0	1	1	0	1	1	0	0	1	1	0	0	0	44
son.14	1	0	0	0	1	0	1	1	1	1	0	0	1	0	1	1	0	0	0	0	48

3. Se creó nueva población repitiendo los pasos anteriores hasta tener una nueva población completa.

3.1. Crossover: la fase más importante de los logaritmos genéticos. Por cada par que se aparean, se elegirá al azar un punto de cruce dentro de los genes.

```
#3.1. Crossover half weight
crossover<- function(child){
  p1<- population[sample(1:500,1),-21]
  p2 <- population[sample(1:500,1),-21]
  (rcut <- which(cumsum(p1*weights_bids[,2])>max_weight/2)[1])
  if(!is.na(rcut)){
    (lcut <- rcut-1)
    proto_child <- c(p1[1:lcut],p2[rcut:20])
  } else {
    (rcut <- which(cumsum(p2*weights_bids[,2])>max_weight/2)[1])
    (lcut <- rcut-1)
    proto_child <- c(p2[1:lcut],p1[rcut:20])
  }
  remove_weight <- which(cumsum(proto_child*weights_bids[,2])>500)
  proto_child[remove_weight] <- 0
  child <- proto_child
  return(child)
}
```

3.2. Mutación: en este caso y problema se generan solamente cuando es necesario. Esta mutación lo que hace es que los genes pueden ser sometidos con una baja probabilidad aleatoria. La mutación se activa cuando identifica que no existe una diversidad dentro la población y con esto previene la convergencia prematura.

```
#3.2. Crear mutación (solo si es necesario)
mutate<- function(child){
  child <- child[2:(length(child)-1)]
  ind<-sample(1:length(child),2)
  i <- child[ind[1]]
  child[ind[1]] <- child[ind[2]]
  child[ind[2]] <- i
  child <- c(1,child,1)
  return(child)
}
```

3.3. Aceptación: el algoritmo termina automáticamente cuando identifica si la población ha convergido. En este paso ha generado varias soluciones al problema.

4. Reemplazo: en este paso genera una nueva población para una nueva serie de algoritmos.

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21
1	0	0	1	0	0	0	1	1	0	1	0	1	0	1	1	0	1	0	1	0	495
2	0	1	0	1	0	0	1	0	0	1	0	1	1	0	1	1	0	0	1	1	493

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21
X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
son	1	1	1	0	0	1	0	1	1	0	0	0	1	1	0	0	1	1	0	1	1
on.1	1	0	1	0	0	0	1	1	0	1	0	0	0	0	0	0	0	1	0	1	1
on.2	1	1	1	1	0	0	1	1	1	0	1	0	0	0	0	1	0	0	0	0	1
on.3	0	1	0	0	0	0	0	0	1	0	1	1	1	0	1	0	1	0	0	0	0
on.4	0	0	0	1	1	0	0	0	1	0	0	1	0	0	0	0	1	1	1	1	0
on.5	1	0	0	0	0	0	1	1	1	0	1	0	0	0	0	0	1	1	0	1	1
on.6	1	0	1	1	1	0	0	0	0	1	1	1	0	1	0	0	1	1	0	0	1
on.7	0	0	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	0	1	1	0

5. Evaluación
6. Loop (ciclo)

En los pasos 5 y 6, estos ocurren simultáneamente, si las condiciones son satisfechas este para, y da como resultado la mejor opción. Y el Loop cuando no se cumple, vuelve a comparar desde el paso 2.

Esto es un ciclo hasta poder conseguir el mejor resultado. Y el resultado para el problema es:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21
on.229	1	1	0	1	1	0	0	0	1	0	0	1	1	0	1	1	1	0	1	0	644
on.202	1	0	1	1	0	1	0	1	1	1	1	0	0	0	0	1	1	0	1	0	637
on.267	1	1	1	1	1	0	0	0	1	1	0	0	0	0	0	1	1	1	1	1	624
on.256	0	0	1	0	0	1	1	0	1	1	0	1	0	1	1	1	1	0	1	1	619
on.240	0	0	1	1	1	1	0	0	1	1	1	0	1	0	1	0	1	0	1	0	616
on.269	1	0	0	1	1	1	1	0	0	1	0	0	1	1	0	1	1	0	1	0	614
on.165	1	1	1	1	0	1	0	0	0	1	1	0	0	0	1	1	0	1	1	0	603
on.273	1	1	0	1	0	1	0	0	0	1	0	1	0	0	1	1	1	0	1	1	603
on.336	1	1	1	1	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	0	595
on.243	0	1	1	0	0	1	1	0	1	1	0	1	0	0	1	1	0	1	1	0	593
on.235	1	0	1	1	0	0	0	1	0	1	0	1	0	0	1	1	1	0	1	0	590
on.316	0	1	0	0	0	1	0	1	1	1	0	0	1	0	1	1	1	0	1	0	589
on.377	1	0	1	1	0	0	0	0	0	1	0	1	1	0	1	1	1	0	1	1	588

Función Anneal

Se utilizaron las funciones vistas en clase para obtener el resultado del problema.

Como primer paso se generó una solución aleatoria.

```
initial_route <- function(n=20,cities_dist){
  dist<-data.frame()
  cities <- nrow(cities_dist)
  for(i in 1:n){
    ruta <- sample_route(1,nrow(cities_dist))
    dist<-rbind( dist, c(ruta, distance_route(ruta,cities_dist) )  )
  }
  names(dist)[ncol(dist)]<-"distance"
  x <- t(dist[dist$distance==min(dist$distance), ])[,1]
  x <- as.integer(x[1:(cities+1) ])
  return(x)
}
```

Como segundo paso se calculo su costo usando alguna función de costo que se haya definido

```
generate_cities <- function(cities =5){
  pos_x <- sample(length(weights_bids[,2]), size = cities, replace = TRUE)
  i <- 1
  weight <- c()
  pay <- c()
  while (i <= cities) {
    weight[i] <- weights_bids[pos_x[i],1]
    pay[i] <- weights_bids[pos_x[i],2]
    i <- i+1
  }
  out <- data.frame(city = 1:cities, weight, pay)
  return(out)
}
```

Como tercer paso se genera una solución vecina aleatoria:

```
rnd_neighbord <- function(vec){
  n <- length(vec)-1
  change <- sample(2:n, size = 2, replace = FALSE)
  temp <- vec[change[1]]
  vec[change[1]] <- vec[change[2]]
  vec[change[2]] <- temp
  return(vec)
}
```

Cuarto paso se Calcula el costo de la nueva solución y quinto paso compara resultados entre la anterior vs la nueva:

```
anneal <- function(cities,N=100, temp=10,    alpha = 0.9){
  temp_min = 0.001

  cities_space <- generate_cities(cities)
  cities_dist <- as.matrix(dist(cities_space[,2:3]))
  ruta <- initial_route(N,cities_dist)
  distancia <- distance_route(ruta,cities_dist)
  while(temp > temp_min){
    i <- 1
    print(temp)
    while(i <= 100){
      new_route <- rnd_neighbord(ruta)
      new_distancia <- distance_route(new_route,cities_dist)
      acceptance_probability <- exp( (distancia-new_distancia)/temp)
      if(acceptance_probability > runif(1) ){
        ruta <- new_route
        distancia <- new_distancia
      } else if(new_distancia<distancia) {
        distancia <- new_distancia
        ruta <- new_route
      }
      i <- i+1
    }
    temp <- temp*alpha
  }
```

Se determinó el costo inicial:

Values	
alpha	num 0.9
cities	5
distancia	130.263720762307
N	100
ruta	int [1:6] 1 2 4 3 5 1
temp	num 10
temp_min	0.001

Se compara con la solución vecina:

Values	
alpha	num 0.9
cities	5
distancia	130.263720762307
i	1
N	100
new_distan...	143.77327703801
new_route	int [1:6] 1 4 2 3 5 1
ruta	int [1:6] 1 2 4 3 5 1
temp	10
temp_min	0.001

En este caso es una solución mejor que la anterior, por lo que se tiene que tomar la decisión de continuar o no.

La simulación de Annealing nos dío como resultado:

sum(cities_space)	double [1]	522

	c.ruta..distancia.
1	1.0000
2	5.0000
3	3.0000
4	4.0000
5	2.0000
6	1.0000
7	130.2637

CONCLUSIONES

- Con el método de logaritmo genéticos se determinó que se tendrá un costo de 644 sin sobrepasar las 500 libras que se tienen restringidas.
- Para obtener los resultados con la función de annealing con este tipo de simulación es un costo de 522 con un peso de 130.

BIBLIOGRAFÍA

- Garduño, R. (12 de Octubre de 2018). *Algoritmos genéticos*. Obtenido de Conogasi: https://conogasi.org/articulos/algoritmos-geneticos/