

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

PROYECTO FINAL
Algoritmos Genéticos y Simulated Annealing

Moisés Eliú Imeri Higueros
21000182

Introducción

El siguiente proyecto tiene como finalidad aplicar los conocimientos recibidos en el curso de Modelación y Simulación I, utilizando una serie de algoritmos que permitan resolver problemas de forma eficiente.

Los algoritmos genéticos consisten en una serie de pasos organizados que describen el proceso que se debe seguir, para dar solución a un problema específico. Este algoritmo hace evolucionar una población de individuos sometiéndola a acciones aleatorias semejantes a las que actúan en la evolución biológica, como por ejemplo las mutaciones y recombinaciones genéticas, así como a una selección de acuerdo con algún criterio en función del cual se decide cuales son los individuos más adaptados, que sobreviven y cuales los menos adaptados los cuales son descartados. Este tipo de algoritmos se enmarcan en los algoritmos evolutivos, que incluyen la programación genética.

Los algoritmos genéticos funcionan entre el conjunto de soluciones de un problema llamado fenotipo y el conjunto de individuos de una población natural, codificando la información de cada solución en una cadena binaria llamada cromosoma. La cadena de símbolos que forman un cromosoma es llamado genotipo. Los cromosomas evolucionan a través de iteraciones, llamadas generaciones, estos son evaluados usando medidas de aptitud, por lo que las siguientes generaciones de cromosomas son generadas aplicando una serie de pasos de **selección, cruzamiento mutación y reemplazo**.

Por otra parte, también existen los algoritmos conocidos como Simulated Annealing, los cuales están basados en el proceso de recocido del acero y cerámicas, una técnica que consiste en calentar y luego enfriar lentamente el material para variar sus propiedades físicas. El calor causa que los átomos aumenten su energía y que puedan así desplazarse de sus posiciones iniciales como un mínimo local de energía, el enfriamiento lento les da mayor probabilidad de recrystalizar en configuraciones con menor energía que la inicial, como un mínimo global.

Normalmente se trabaja con vectores de configuración, energía del sistema, temperatura del sistema y diferencias de temperatura entre los vectores de configuración. Para poder crear este algoritmo es **necesario generar una solución aleatoria, seguidamente calcular el costo usando una función de costo, generar una solución aleatoria vecina, calcular el nuevo costo de solución y finalmente compararlos**, hasta definir y establecer la solución óptima.

A continuación, utilizando los conocimientos adquiridos en el curso de Modelación y Simulación, se presenta la solución de un problema utilizando algoritmos genéticos y Simulated Annealing, describiendo cada uno de sus pasos así como los resultados obtenidos.

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 restringida por la capacidad de carga del avión. La compañía aérea tiene que seleccionar un subconjunto de paquetes que su peso total sea menor al 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.

Entregables

1. Solución utilizando algoritmos genéticos
2. Solución utilizando simulated annealing.
3. Documento donde explica cada una de las soluciones con capturas de pantalla de las salidas de su código.

Recomendaciones

En este problema los vectores son binarios es decir que si tenemos un 1 en la posición 5 significa que vamos a transportar el paquete 5.

1 0 1 0 1 = llevamos el paquete 1,3,5

0 0 1 1 0 = llevamos el paquete 3, 4

El objetivo (Fitness) es la ganancia, solo tenemos la restricción de que no podemos llevar más de 500 de peso.

Solución: Algoritmo Genético

1. **[START]** Se genera una población aleatoria con n cromosomas
 - Se inició creando una matriz que contiene el peso y el costo de los 20 paquetes.

	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

- Se creó una función que permita llevar la sumatoria de las libras para no pasar la restricción de 500 lb. Así como el conteo de paquetes para que no pueda repetirse la selección de estos, por lo que se generan cromosomas que cumplan con la restricción.

Values	
i	12L
out	num [1:20] 0 1 0 1 0 0 1 1 0 1 ...
pool	num [1:20] 55 31 43 23 48 40 53 54 39 77 ...
pool_idx	int [1:8] 1 3 6 9 15 16 17 19
selected	int [1:11] 8 13 7 4 10 2 14 20 11 12 ...
value_idx	5L
weight	537
weights	num [1:20] 55 31 43 23 48 40 53 54 39 77 ...

- Esta función va seleccionando paquetes de pool, acumulando el peso hasta llegar a 500 libras, una vez se llega al límite de 500lbs, se genera un out el cual selecciona las posiciones de los paquetes seleccionados. Por medio de esta función se realiza la selección de los cromosomas

Values	
i	12L
out	num [1:20] 0 1 0 1 0 0 1 1 0 1 ...
pool	num [1:20] 55 31 43 23 48 40 53 54 39 77 ...
pool_idx	int [1:8] 1 3 6 9 15 16 17 19
selected	int [1:11] 8 13 7 4 10 2 14 20 11 12 ...
value_idx	5L
weight	537
weights	num [1:20] 55 31 43 23 48 40 53 54 39 77 ...

- Se genera una matriz de cromosomas para determinar que cromosoma es el que maximiza el problema, según la simulación realizada el cromosoma con mayores ganancias es el #8, con un costo de 649

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21	
8	1	0	1	1	0	1	0	1	1	0	1	1	0	1	0	1	0	1	1	0	649	
360	1	0	1	1	1	1	0	0	1	0	1	0	1	1	0	1	0	1	1	0	649	
447	1	1	0	1	1	1	0	1	0	0	0	0	0	1	1	1	1	0	1	0	631	
225	1	0	1	1	0	1	0	1	1	0	0	1	0	0	0	1	1	1	1	0	630	
86	0	0	0	1	0	1	0	1	1	1	0	0	0	1	1	1	1	0	1	0	623	
451	0	1	0	1	0	1	0	1	1	1	0	0	0	1	0	1	1	1	1	0	620	
118	1	0	1	1	1	1	0	0	1	0	1	1	1	1	0	1	0	1	0	0	618	
364	1	0	1	1	1	0	0	1	1	0	0	0	1	1	1	0	0	1	1	0	615	
17	1	1	1	0	1	1	0	0	1	1	0	0	0	1	1	1	0	1	0	0	610	

Con esta serie de pasos ya se cuenta con una población inicial para poder realizar la nueva población.

2. [FITNESS] Se evalúa el fitness $f(x)$ de cada cromosoma en la población

- Se crea una variable que medirá el FITNESS, en este caso medirá el pago por envío de los paquetes, se busca encontrar los cromosomas con el mayor FITNESS ya que es un problema de maximización, en el cual se busca la mayor ganancia posible.

```
population <- t(sapply(1:500,function(x,w){chromosome(w)},w=weights_bids[,1] ))
population <- cbind(population,population %*% weights_bids[,2])
```

3. **[NEW POPULATION]** Se crea una nueva población repitiendo una serie de pasos hasta que la nueva población esté completa

CROSSOVER

- Por medio de una probabilidad crossover se generan dos padres para generar una descendencia (CHILDREN).
- Se generan dos padres utilizando cromosomas aleatorios de 1 a 500 de la población inicial.

```
p1<- population[sample(1:500,1),-21]
p2 <- population[sample(1:500,1),-21]
> p1
[1] 0 1 0 0 1 0 0 1 1 1 1 0 0 0 0 1 0 1 1 0
> p2
[1] 0 0 1 1 1 0 0 1 0 1 0 1 0 0 0 0 1 1 1 0
> |
```

- Para crear a la descendencia (CHILD) se utilizará la mitad de cada padre, para ello se coloca una restricción que seleccione del padre 1 la cantidad de paquetes que sumen hasta 250 libras y del padre 2 se tomara otra mitad de paquetes que sumados den hasta 250 libras, formando un CHILD que cumple con las restricción de peso de 500 libras.

```
> proto_child
[1] 0 1 0 0 1 0 0 1 1 1 0 1 0 0 0 0 1 1 1 0
> |
```

- Por medio de la función crossover se logra determinar al cromosoma hijo, formado a partir de dos padres de la población inicial.

```
> p1
[1] 0 1 0 0 1 0 0 1 1 1 1 0 0 0 0 0 1 0 1 1 0
> p2
[1] 0 0 1 1 1 0 0 1 0 1 0 1 0 0 0 0 1 1 1 0
> child
[1] 0 1 0 0 1 0 0 1 1 1 0 1 0 0 0 0 1 1 1 0
> |
```

Mutation: Por medio de una probabilidad de mutación, de ser necesario, se realiza una nueva descendencia intercambiando posiciones de los genes dentro de los cromosomas hijos.

Values	
child	num [1:20] 1 1 0 1 1 1 0 0 1 0 ...
i	1
index	int [1:2] 6 3

```

Traceback
Show internals
→ mutacion(crossover(test))

```

- Al ejecutar la función MUTACION se realizan los cambios de la variable CHILD seleccionando las posiciones 6 y 3 y haciendo el intercambio respectivo.
- Esta mutación solo se realizará si se cumple la condición de que la probabilidad de la variable child sea menor a 0.01.

NEW POPULATION

- Una vez se han creado los cromosomas padres y los cromosomas hijos, se procede a crear una nueva población con los cromosomas hijos.
- Se genera una nueva matriz partiendo de los cromosomas hijos obteniendo el siguiente dataframe con la nueva población.

	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
hijo	0	0	0	1	0	1	1	1	0	0	0	1	1	1	0	1	1	0	0	1	0
hijo.1	0	0	0	1	1	0	1	0	0	1	1	0	0	1	0	0	1	0	1	1	0
hijo.2	1	0	0	1	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	1	1
hijo.3	0	1	0	1	0	1	1	1	0	0	1	1	0	1	1	0	0	0	0	0	0
hijo.4	0	1	0	1	0	0	0	1	1	0	0	0	0	0	1	0	0	1	0	1	0
hijo.5	0	0	1	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	1	0
hijo.6	0	0	0	1	1	0	0	1	1	0	0	0	1	0	0	0	1	0	1	1	0
hijo.7	0	1	0	0	0	1	0	1	1	0	1	0	1	0	0	0	1	1	0	0	0
hijo.8	1	0	0	0	0	1	1	0	1	1	0	0	0	1	1	0	1	0	1	0	1
hijo.9	1	1	1	0	0	0	1	0	0	1	0	0	0	0	1	0	1	0	1	0	1
hijo.10	1	1	0	0	0	0	0	0	1	1	0	1	0	0	0	1	1	1	0	0	1
hijo.11	0	0	1	0	0	0	0	1	1	1	0	0	0	1	0	1	0	0	1	0	0
hijo.12	0	0	0	0	0	0	1	1	1	1	1	0	1	0	0	0	1	1	1	1	0
hijo.13	0	1	1	1	0	0	1	1	1	0	0	0	0	1	1	1	0	0	0	0	0
hijo.14	1	0	0	1	0	1	0	0	0	0	1	1	1	0	0	0	1	0	0	1	1
hijo.15	0	1	0	1	0	0	1	0	0	1	0	0	1	0	0	1	1	0	0	1	0
hijo.16	1	1	0	1	0	0	1	0	1	1	0	0	0	1	1	0	0	0	0	0	1
hijo.17	1	1	0	0	1	0	1	0	0	1	0	0	1	0	1	1	1	0	0	0	1
hijo.18	0	0	0	0	0	0	0	1	0	1	0	1	0	1	1	0	1	0	1	0	0
hijo.19	1	1	0	1	1	1	1	1	0	0	0	0	1	0	1	0	0	0	0	0	1
hijo.20	1	0	1	0	1	0	0	1	0	0	1	0	1	0	1	1	1	0	0	0	1
hijo.21	1	1	1	0	0	0	1	1	0	1	0	0	1	0	0	0	1	0	1	1	1
hijo.22	0	0	1	0	1	0	0	0	1	0	0	1	1	1	1	1	0	0	0	0	0
hijo.23	1	1	0	0	1	0	0	0	1	0	0	1	1	1	1	0	0	0	0	1	1
hijo.24	0	0	0	0	0	1	1	0	1	1	0	1	1	1	0	1	0	0	0	0	0
hijo.25	1	0	0	1	0	0	1	1	0	0	0	0	0	1	0	1	0	1	0	0	1
hijo.26	1	0	1	1	0	1	0	1	1	1	0	0	1	0	0	0	0	0	0	0	1
hijo.27	1	1	0	0	1	1	0	0	0	1	0	1	1	0	0	1	1	0	0	1	1
hijo.28	1	1	1	0	0	0	1	0	1	0	0	1	1	0	0	0	1	0	1	0	1
hijo.29	1	1	0	1	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	1
hijo.30	0	0	0	1	1	0	0	1	1	0	1	1	1	1	1	0	0	0	0	0	0
hijo.31	1	0	0	0	1	1	0	1	0	1	1	1	0	0	1	1	0	0	0	0	1

4. [REPLACE] Utilizando la nueva población se realizan nuevamente las corridas del algoritmo

- Una vez generada la nueva población se procede a realizar una nueva simulación, para obtener el CHILD con el mejor fitness.
- Se obtiene una nueva matriz con los CHILD

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21
hijo	0	0	1	0	1	1	1	0	0	0	1	1	1	1	0	1	1	0	1	0	490
hijo.1	0	0	1	1	0	1	0	0	1	1	0	0	0	1	0	0	1	1	0	0	342
hijo.2	0	0	1	0	0	1	0	1	0	1	0	0	1	0	0	0	0	0	1	1	286
hijo.3	1	0	1	0	1	1	1	0	0	1	1	0	1	1	0	0	0	0	0	0	436
hijo.4	1	0	1	0	0	0	1	1	0	0	0	0	0	1	0	0	1	0	1	0	343
hijo.5	0	1	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	1	0	315
hijo.6	0	0	1	1	0	0	1	1	0	0	0	1	0	0	0	1	0	1	1	0	346
hijo.7	1	0	0	0	1	0	1	1	0	1	0	1	0	0	0	1	1	0	0	0	444
hijo.8	0	0	0	0	1	1	0	1	1	0	0	0	0	1	1	0	1	0	1	1	432
hijo.9	1	1	0	0	0	1	0	0	1	0	0	0	0	1	0	1	0	1	0	1	301
hijo.10	1	0	0	0	0	0	0	1	1	0	1	0	0	0	1	1	1	0	0	1	398
hijo.11	0	1	0	0	0	0	1	1	1	0	0	0	1	0	1	0	0	1	0	0	331
hijo.12	0	0	0	0	0	1	1	1	1	1	0	1	0	0	0	1	1	1	1	0	498
hijo.13	1	1	1	0	0	1	1	1	0	0	0	0	1	1	1	0	0	0	0	0	428
hijo.14	0	0	1	0	1	0	0	0	0	1	1	1	0	0	0	1	0	0	1	1	392
hijo.15	1	0	1	0	0	1	0	0	1	0	0	1	0	0	1	1	0	0	1	0	391
hijo.16	1	0	1	0	0	1	0	1	1	0	0	0	0	1	1	0	0	0	0	1	372
hijo.17	1	0	0	1	0	1	0	0	1	0	0	1	0	1	1	1	0	0	0	1	393
hijo.18	0	0	0	0	0	0	1	0	1	0	1	0	1	1	0	1	0	1	0	0	278
hijo.19	1	0	1	1	1	1	1	0	0	0	0	1	0	1	0	1	0	0	0	1	391

5. [TEST] Si la condición final es satisfactoria el algoritmo se detiene y brinda la solución óptima

- Al realizar la simulación se obtiene una nueva matriz la cual refleja los resultados óptimos, ordenados del hijo con mayores ganancias al hijo con menores ganancias, esto permite seleccionar la solución óptima del problema.

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21
hijo.213	1	1	0	0	0	1	1	1	0	1	1	1	1	1	0	0	1	0	1	0	599
hijo.179	1	1	0	0	1	1	1	0	0	1	1	1	1	0	1	1	0	0	0	0	562
hijo.387	1	1	1	0	0	1	1	0	0	1	1	0	1	1	1	0	1	0	0	0	552
hijo.410	1	1	0	0	1	1	1	0	1	1	1	1	0	0	1	1	0	0	0	0	548
hijo.53	1	0	0	1	0	1	1	0	0	1	1	0	0	0	1	1	1	0	1	1	546
hijo.390	0	1	1	1	0	1	0	0	0	1	0	1	1	0	1	1	1	0	1	0	543
hijo.292	1	0	0	1	1	0	1	0	0	1	1	0	0	0	1	1	1	1	1	0	542
hijo.237	0	1	0	0	1	1	0	0	0	1	1	1	0	0	1	1	1	1	1	0	541
hijo.196	0	0	0	1	1	1	1	1	0	1	0	1	1	0	1	0	1	0	0	0	540
hijo.455	1	1	0	0	0	1	1	0	1	1	0	0	1	0	1	0	0	1	1	1	539

6. [LOOP] Genera nuevamente el algoritmo hasta encontrar la solución óptima

- La función de nueva población genera un LOOP en búsqueda de la maximización del problema y se detiene hasta obtener la de mejores resultados.

Resultado: Según la simulación realizada por medio de algoritmo genético, se determinó que la solución óptima es la del cromosoma hijo 213, con la cual se obtiene un total de Q599, cumpliendo con la restricción de peso, seleccionando los paquetes 1,2,6,7,8,10,11,12,13,14,17 y 19.

Solución SIMULATED ANNEALING

Con base en las funciones vistas en clase se desarrolló el siguiente código:



1. En primer lugar, generamos una solución aleatoria del problema
 - Se genera una población de paquetes, para ello se realizó la función `generated_paquetes`

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

- Se utilizó una función “`initial_paquete`” la cual plantea una solución aleatoria para encontrar el punto óptimo.

```
initial_paquete <- function(n=20,pac_costo){  
  dist<-data.frame()  
  paquete <- nrow(pac_costo)  
  for(i in 1:n){  
    ruta <- sample_paquete(1,nrow(pac_costo))  
    dist<-rbind( dist, c(ruta, costo_paquete(ruta,pac_costo) ) )  
  }  
  names(dist)[ncol(dist)]<-"costo"  
  x <- t(dist[dist$costo==min(dist$costo), ])[,1]  
  x <- as.integer(x[1:(paquete+1) ])  
  return(x)  
}
```

2. Calculamos el fitness o el costo de la solución, para ello utilizamos una función que calcule el costo total de la selección de paquetes.
 - Se utiliza un total de 50 simulaciones para calcular la solución óptima, con una temperatura de 20 y un Alpha de 0.95, según la tabla se observa que la solución aleatoria tiene un costo de 476.99

Data	
▶ capacidad	10 obs. of 3 variables 
pac_costo	num [1:10, 1:10] 0 17 49.... 
Values	
alpha	0.95
costo	476.989536793543
i	1
N	50

```

anneal <- function(paquetes=5, N=50, temp=20, alpha = 0.95){
  temp_min = 0.001

  capacidad <- generated_paquetes(paquetes)
  pac_costo <- as.matrix(dist(capacidad[,2:3]))
  ruta <- initial_paquete(N,pac_costo)
  costo <- costo_paquete(ruta,pac_costo)
  while(temp > temp_min){
    i <- 1
    print(temp)
    while(i <= 100){
      nuevos_paquetes <- vecino(ruta)
      nuevo_costo <- costo_paquete(nuevos_paquetes,pac_costo)
      acceptance_probability <- exp( (costo-nuevo_costo)/temp)
      if(acceptance_probability < runif(1) ){
        ruta <- nuevos_paquetes
        costo <- nuevo_costo
      } else if(nuevo_costo>costo) {
        costo <- nuevo_costo
        ruta <- nuevos_paquetes
      }
      i <- i+1
    }
    temp <- temp*alpha
  }
}

```

3. Seguidamente creamos un vector vecino, el cual tiene pequeñas modificaciones de la posición del vector inicial. Para hacer esto utilizamos una función de swap.

```

vecino <- 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)
}

```

- Se genera otra solución aleatoria vecina con la cual se tienen los siguientes parámetros:

Data	
capacidad	10 obs. of 3 variables
pac_costo	num [1:10, 1:10] 0 17 49...
Values	
alpha	0.95
costo	476.989536793543
i	1
N	50
nuevo_costo	492.742942873979
nuevos_paq...	int [1:11] 1 10 5 8 4 2 7 3...
paquetes	10
ruta	int [1:11] 1 5 10 8 4 2 7 3...
temp	20
temp_min	0.001

- Se observa que el nuevo costo es de 492.74 por lo que la solución vecina tiene una solución más eficiente.
 - En el momento que el vecino tenga un mejor costo, este costo se convertirá en el nuevo costo inicial para la siguiente iteración.
 - Aún se tiene una temperatura de 20
4. Calculamos el costo de la nueva solución, esto para poder comparar las dos soluciones planteadas y escoger aquella que maximice el problema.
- Al terminar las iteraciones y la temperatura llegar a ser menor a 0.001 lo cual ayuda a que el algoritmo sea más preciso en cada iteración, se obtuvieron los siguientes resultados:
 - **Se llevarán los paquetes 3,10,2,9,4,5,8,7,6 con un peso total de 390lbs por un precio de Q432.00. Con esto se llega a la solución óptima para maximizar el problema**

	c.ruta..costo.	
1	1.0000	[1] 0.001857815
2	3.0000	[1] 0.001764925
3 2	10.0000	[1] 0.001676678
4	2.0000	[1] 0.001592844
5	9.0000	[1] 0.001513202
6	4.0000	[1] 0.001437542
7	5.0000	[1] 0.001365665
8	8.0000	[1] 0.001297382
9	7.0000	[1] 0.001232513
10	6.0000	[1] 0.001170887
11	1.0000	[1] 0.001112343
12	390.0013	[1] 0.001056726
		[1] 0.001003889
		[1] 432

```

    comp ~ comp alpha
  }
  print(sum(capacidad[,3]))
  plot(capacidad[,2:3])
  x<- ruta
  polygon(capacidad[x,2:3],border="red",lwd=3)
  return(View(data.frame(c(ruta, costo))))
}

```

