

Proyecto Final

En la realización del proyecto se define el vector de los pesos que se tiene para enviar al avión en la cual se realiza una matriz con dichos valores que son las siguientes:

	weight	pay
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	59	78
10	77	65
11	49	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

Para crear el algoritmo genético se utilizó códigos que se vio en clases y dado también por el ingeniero se realiza los cromosomas y la población de la siguiente forma:

```
> chromosome(weights_pay[,2])  
[1] 1 0 0 0 1 1 1 0 1 1 0 1 0 1 0 0 0 0 0 0
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21
1	1	1	0	0	0	0	0	0	1	1	0	0	1	0	1	1	0	1	0	0	412
2	0	1	1	0	1	1	0	1	1	1	0	0	1	0	0	0	0	1	0	1	477
3	0	0	1	1	0	1	0	0	1	1	1	0	0	1	0	0	1	0	1	1	471
4	0	1	1	0	0	1	0	1	0	0	1	1	1	1	1	0	0	1	0	1	500
5	1	0	0	0	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0	1	464
6	0	0	1	0	0	1	1	1	1	1	1	0	1	0	0	1	0	1	0	0	494
7	0	0	0	1	1	1	0	1	1	0	1	1	0	1	1	0	0	0	0	1	473
8	1	0	0	0	0	1	0	1	1	0	1	0	0	0	1	0	0	1	1	0	406
9	1	0	0	0	0	1	0	1	0	0	0	1	0	1	0	1	0	1	1	1	392
10	0	1	0	0	0	1	1	1	1	1	1	0	0	0	0	0	1	0	1	0	473
11	1	0	0	1	0	1	1	1	0	1	0	0	0	1	1	0	0	0	1	1	401
12	1	0	0	0	1	1	0	1	0	1	0	0	1	0	0	0	1	0	1	1	479
13	0	0	0	1	0	1	1	1	1	0	0	1	1	0	0	0	1	1	0	1	475
14	1	1	1	1	1	1	0	1	0	0	0	0	0	0	1	1	0	0	0	0	401

Obviamente en este dato ya nos dan el fitness en la casilla 21 para que se pueda trabajar.

Por lo que se realiza el crossover que nos proporciona el ingeniero en conjunto se realiza la parte de la mutación pero tomando en cuenta para que pueda mutar debe de estar debajo de 0.01 y se realiza dicha mutación

Como se muestra los siguientes códigos.

```
##### crossover half weight
crossover <- function(child){
  p1<- population[sample(1:500,1),-11]
  p2 <- population[sample(1:500,1),-11]
  (rcut <- which(cumsum(p1*weights_pay[,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_pay[,2])>max_weight/2)[1])
    lcut <- rcut-1
    proto_child <- c(p2[1:lcut],p1[rcut:20])
  }
  remove_weight <- which(cumsum(proto_child*weights_pay[,2])>500)
  proto_child[remove_weight] <- 0
  child <- proto_child
  return(child)
}

#Mutate
mutar <- function(child){
  child <- child[2:(length(child)-1)]
  index<-sample(1:length(child),2)
  i <- child[index[1]]
  child[index[1]] <- child[index[2]]
  child[index[2]] <- i
  child <- c(1,child,1)
  return(child)
}
```

Ya realizado en esta parte se realiza el código principal para poder generar una nueva población y poder encontrar el fitness optimo del problema como se muestra a continuación:

```

N = 20
new_pop <- matrix(rep(0,N=1),ncol = N+1)
for (i in 1:500) {
  parents <- sample(1:nrow(population),size = 2)
  padres <- population[parents,1:(N+1)]
  hijo<-crossover(padres)
  if(runif(1)<0.01){
    hijo<-mutar(hijo)
  }
  new_pop <- rbind(new_pop,hijo)
}
population <- new_pop[-1,-1]
population <- cbind(population,population %%% weights_pay[,1])

generate_cities <- function(cities =5){
  pos_y <- sample(weights_pay[,2], size = cities , replace=TRUE)
  out <- data.frame(city = 1:cities, pos_y )
  return(out)
}

```

Resultado

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21
hijo	1	0	1	0	0	1	1	1	1	1	0	1	1	1	0	0	0	0	0	0	521
hijo.1	0	0	0	1	0	0	0	0	1	0	1	1	1	0	0	1	0	0	0	1	321
hijo.2	0	0	1	1	0	1	1	1	0	0	0	1	0	0	1	1	0	0	0	0	379
hijo.3	1	1	1	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	0	1	386
hijo.4	0	1	1	1	1	1	0	0	1	1	0	1	1	0	0	0	0	0	0	1	475
hijo.5	0	1	1	1	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	1	286
hijo.6	0	1	0	0	1	1	1	0	1	1	0	1	1	0	1	0	0	0	0	1	533
hijo.7	1	0	1	0	0	1	1	0	0	1	1	0	0	0	1	1	0	0	0	0	424
hijo.8	1	0	0	1	1	0	1	0	0	0	0	1	1	0	0	1	0	1	0	0	357
hijo.9	1	0	0	0	1	1	1	0	1	1	0	1	0	1	0	1	0	0	0	1	497
hijo.10	0	1	0	1	0	0	1	0	0	1	1	1	0	0	1	1	0	1	0	0	429
hijo.11	0	0	0	0	1	1	0	0	0	1	1	0	1	0	0	1	1	0	0	1	407
hijo.12	0	1	0	1	1	0	0	0	0	1	1	1	1	0	1	1	0	0	0	1	489
hijo.13	1	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	0	1	0	1	308
hijo.14	0	0	0	0	0	0	1	0	0	1	1	0	1	0	1	1	0	0	0	1	381
hijo.15	0	0	0	1	1	1	0	1	0	1	1	0	0	1	0	0	1	0	0	0	381
hijo.16	1	1	0	0	1	0	0	0	1	0	1	0	1	0	0	0	1	0	0	0	357
hijo.17	1	0	0	1	0	0	1	0	1	0	1	1	0	1	1	0	1	0	0	0	459
hijo.18	1	0	1	0	1	0	0	0	1	1	0	0	1	1	0	1	1	0	0	0	461
hijo.19	1	1	0	1	1	0	0	0	0	1	0	1	0	1	1	1	1	0	0	0	490
hijo.20	0	1	1	1	0	1	1	0	0	1	0	0	0	0	0	0	1	0	0	0	329

Eso es el resultado final del algoritmo genético, ahora con la realización del simulated annealing se realiza de la siguiente forma utilizando distancias, rutas, generación de ciudades y vecinos que son las siguientes:

```

generate_cities <- function(cities =5){
  pos_y <- sample(length(weights_pay[,2]), size = cities , replace=TRUE)
  weight <- c()
  pay<- c()
  for(i in 1:length(pos_y)){
    weight[i]<-weights_pay[pos_y[i],1]
    pay[i]<- weights_pay[pos_y[i],2]
  }
  out <- data.frame(city = 1:cities,weight,pay)
  return(out)
}

distance_route <- function(route,distance){
  sum_distance<-0
  for(i in 1:length(route[-1])){
    out <- distance[ route[i], route[i+1] ]
    sum_distance <- sum_distance + out
  }
  return(sum_distance)
}

sample_route <- function(initial_node = 1, nodes = 4){
  nodes <- 1:nodes
  sample_nodes <- nodes[-initial_node]
  attach_nodes <- sample(sample_nodes)
  out <- c(initial_node,attach_nodes,initial_node)
  return(out)
}

initial_route <- function(n=200,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)
}

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)
}

```

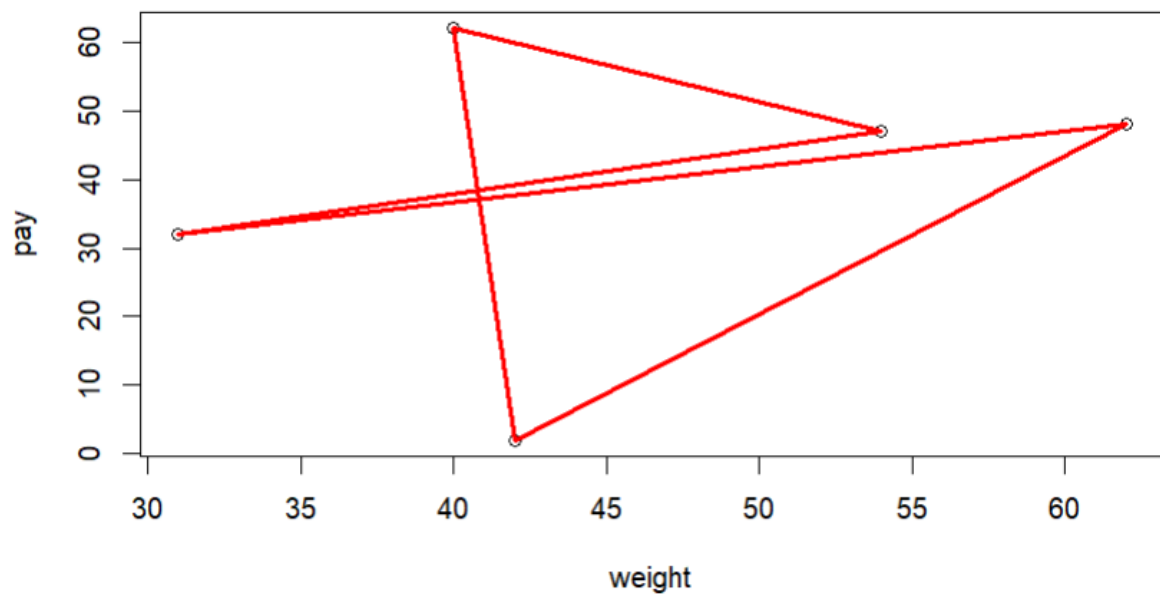
En la cual cada uno de las funciones realiza la distancia y la ruta mas corta posible para realizar este metodo utilizando con el codigo principal:

```

anneal <- function(cities=5,N=300, temp=10, alpha = 0.9){
  repeat{
    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(1,0,1) ){
          ruta <- new_route
          distancia <- new_distancia
        } else if(new_distancia>distancia) {
          distancia <- new_distancia
          ruta <- new_route
        }
        i <- i+1
      }
      temp <- temp*alpha
    }
    if(sum(cities_space[,2])<=500){
      print(sum(cities_space[,3]))
      plot(cities_space[,2:3])
      x<- ruta
      polygon(cities_space[x,2:3],border="red",lwd=3)
      return(View(data.frame(c(ruta,distancia))))
      break
    }
  }
}

```



Dando como resultado lo siguiente:



Teniendo un costo de 191

```
[1] 191
```

Y teniendo la siguiente ruta con un peso de 193.05kg

	c.ruta..distancia. 
1	1.0000
2	2.0000
3	5.0000
4	4.0000
5	3.0000
6	1.0000
7	193.0559