Practica Computación Evolutiva

Erika Ortiz

2023-02-15

Objetivo

- El objetivo es poner en una fila a 100 personas que llamaremos P1 a P100.

Restricciones

- La persona Pi no puede estar a i personas de distancia de la P(i+1). Esta restricción es válida para i desde 1 hasta 99, la P100 no tiene restricciones, solo la que le viene de P99.

Ejemplos:

- La persona P5 no puede estar a 5 personas de la persona 6.
- La persona P1 no puede estar a distancia 1 de la P2.
- Como ejemplo con 5 personas, una disposición correcta podría ser: P1,P5,P3,P2,P4 porque la P1 no está a distancia 1 de la P2 (está a distancia 3), la P2 no está a distancia 2 de P3 (está a distancia 1), la P3 no está a distancia 3 de P4 (está a distancia 2), la P4 no está a distancia 4 de P5 (está a distancia 3).
- Un ejemplo de una fila incorrecta sería: P1,P3,P5,P2,P4 porque P3 está a distancia 3 de P4.
- Una vez lo tengáis desarrollado podéis tomar el número de personas como un parámetro que podamos cambiar.

Función Fitness

En este problema, se asume que cada elemento de la permutación representa una persona en una cola, y el número de la posición de cada persona en la cola determina la distancia que tiene que mantener con respecto a las personas adyacentes. Por ejemplo, la persona en la posición 1 de la cola no puede estar a una distancia de una persona en la posición 2.

La función **fitness_function** recibe como entrada una permutación de n_people elementos, donde cada elemento es una posición en la cola. La función calcula el número de colisiones que existen en la permutación, es decir, el número de veces que una persona viola la distancia que debe mantener con respecto a las personas adyacentes. La función itera sobre la permutación y compara la distancia entre cada pareja de personas adyacentes. Si la distancia es igual a la posición relativa de las personas en la cola, se cuenta como una colisión.

Finalmente, la función retorna el número total de colisiones en la permutación.

```
fitness_function <- function(population) {
  n_people <- length(population)
  collisions <- 0
  for (i in 1:(n_people - 1)) {
    distance <- abs(population[i] - population[i + 1])
    if (distance == i) {
      collisions <- collisions + 1
    }
```

```
  }
  return(collisions)
}
```

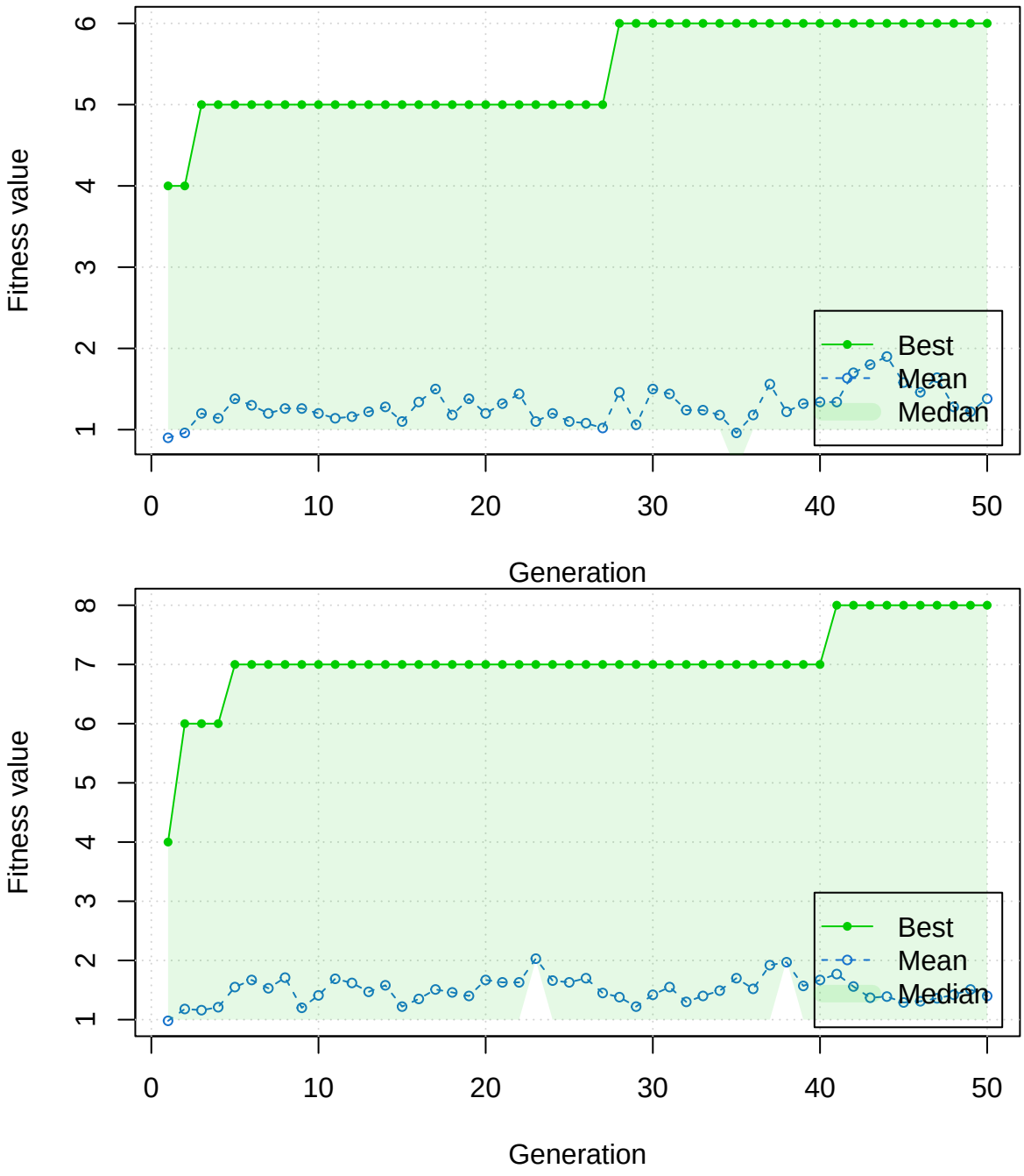
Función GA

La función GA contiene un un bucle que itera sobre diferentes tamaños de población, ejecutando un algoritmo genético para resolver un problema de optimización de permutación.

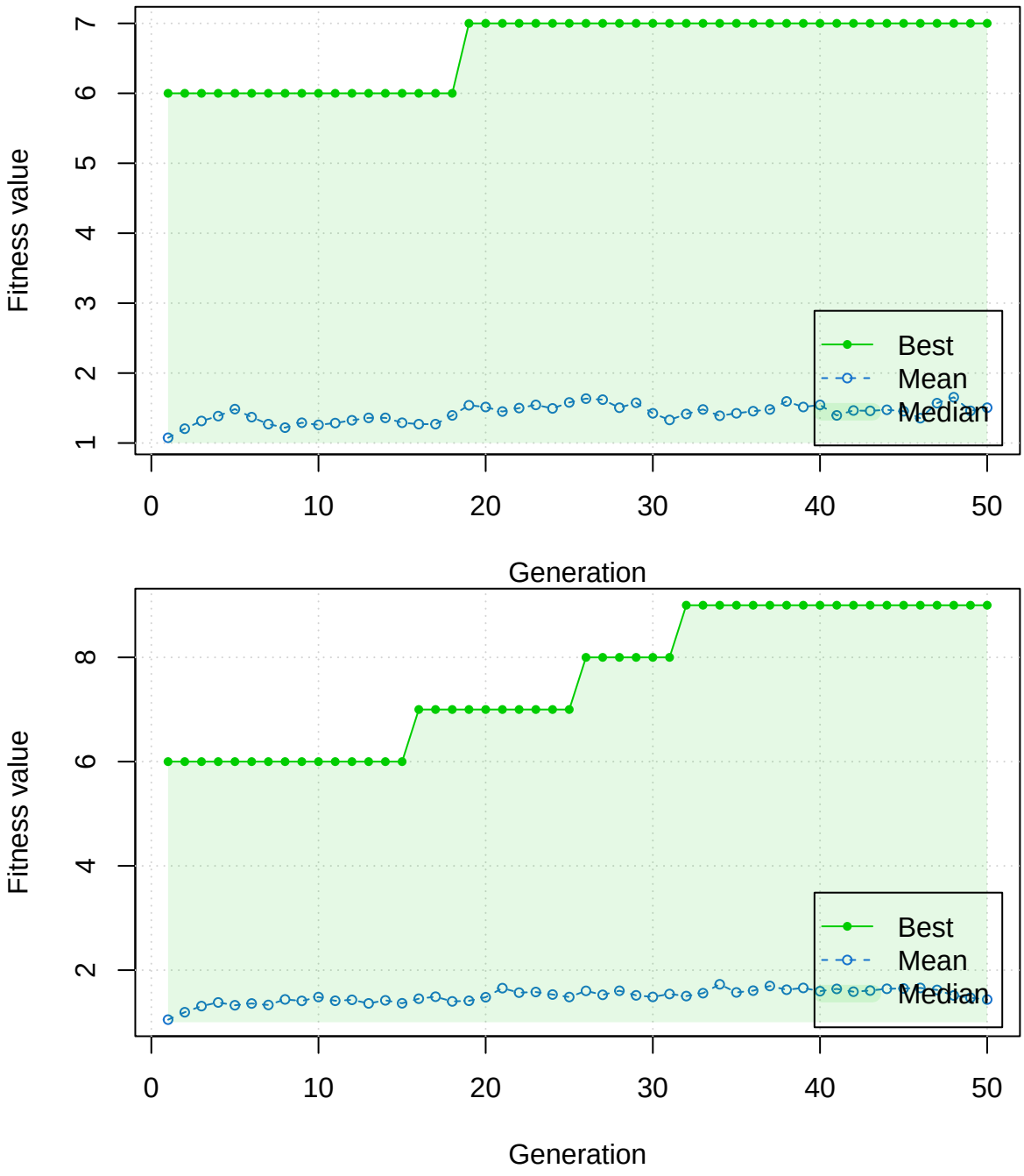
En cada iteración del bucle, se establece una semilla aleatoria para garantizar que los resultados sean reproducibles y se ejecuta el algoritmo genético con los parámetros de configuración especificados, incluyendo el tamaño de la población, el número máximo de iteraciones, la tasa de mutación y la tasa de cruce.

Al final del bucle, el vector "collisions" contiene los valores de la función de fitness para la mejor solución encontrada para cada tamaño de población. Estos valores se pueden utilizar para comparar el rendimiento del algoritmo genético para diferentes tamaños de población. El bucle se repite "length(pop_sizes)" veces, es decir, para cada tamaño de población especificado en el vector "pop_sizes".

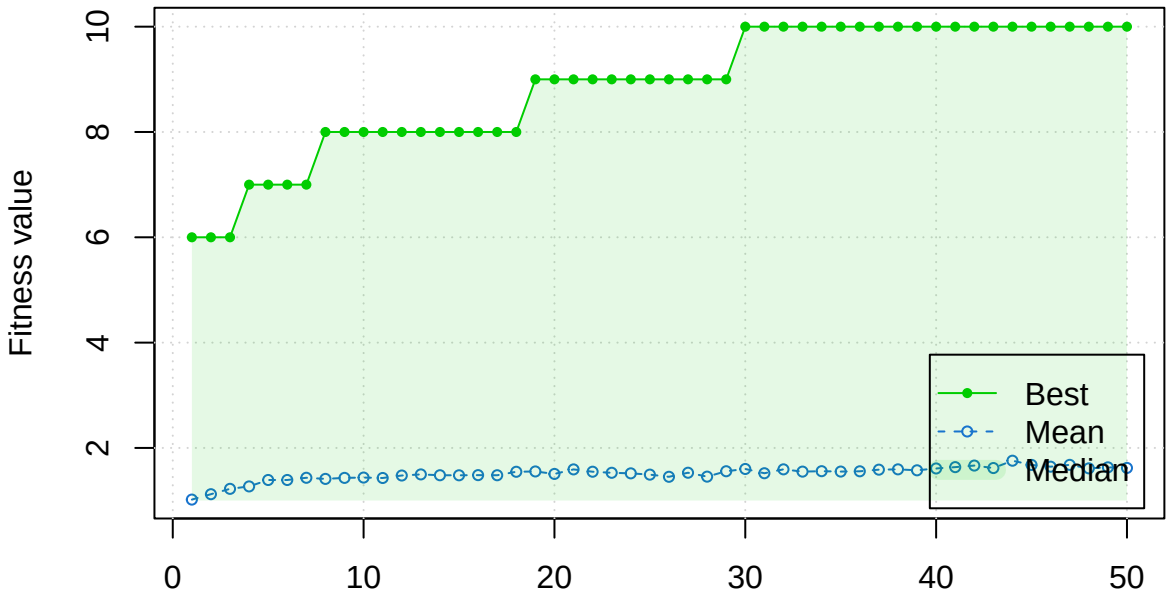
```
pop_sizes <- c(50, 100, 200, 500, 1000, 2000)
collisions <- numeric(length(pop_sizes))
max_iter = 50
for (i in 1:length(pop_sizes)) {
  set.seed(1)
  GA <- ga(
    type = "permutation",
    maxiter = max_iter,
    fitness = fitness_function,
    lower = 1,
    upper = n_people,
    popSize = pop_sizes[i],
    pmutation = 0.5,
    pcrossover = 0.8
  )
  if (nrow(GA@solution) > 0) {
    summary(GA)
    plot_pop = plot(GA)
    best_solution <- GA@solution[1,]
    collisions[i] <- fitness_function(best_solution)
  } else {
    collisions[i] <- NA
  }
}
```

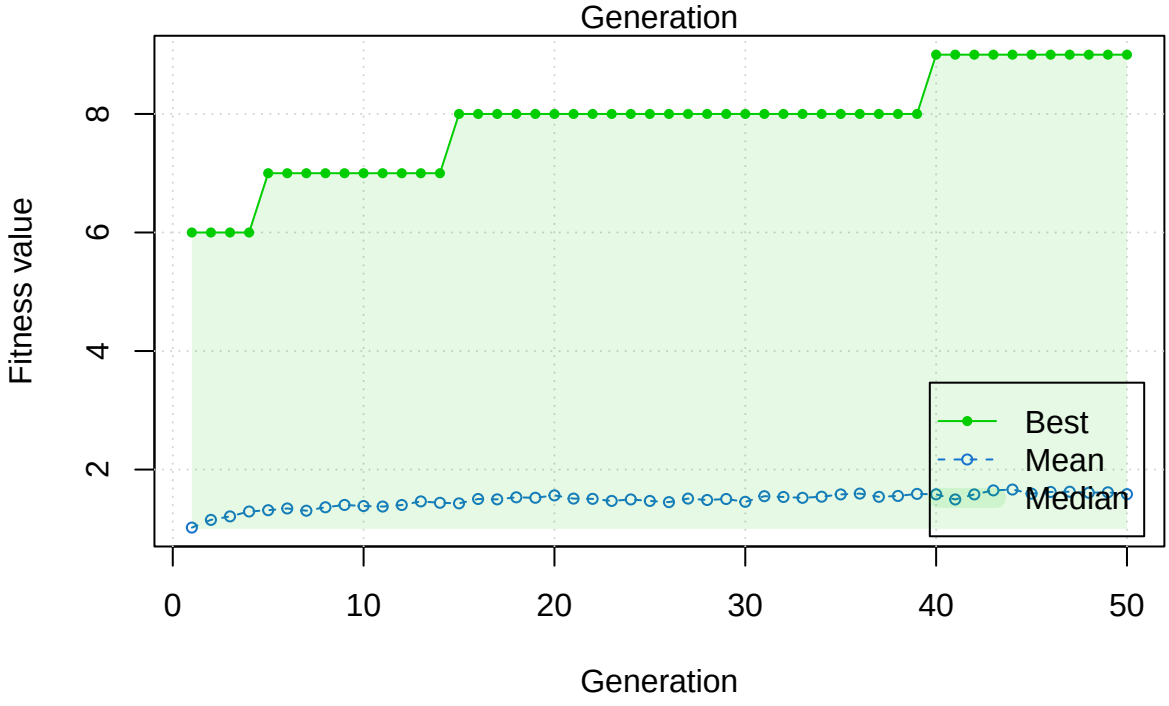

Fitness value
1
2
3
4
5
6
0
10
20
30
40
50
Generation
Best
Mean
Median

Fitness value
1
2
3
4
5
6
7
8
0
10
20
30
40
50
Generation
Best
Mean
Median


Fitness value
1
2
3
4
5
6
7
0
10
20
30
40
50
Generation
Best
Mean
Median

Fitness value
2
4
6
8
0
10
20
30
40
50
Generation
Best
Mean
Median





Resultados

Este código crea una tabla de resultados de las colisiones mínimas para cada tamaño de población utilizado en el bucle anterior.

- La columna "tamaño_poblacion" contiene los diferentes tamaños de población utilizados que corresponden a pop_sizes <- c(50, 100, 200, 500, 1000, 2000).
- La columna num_min_choques: contiene el valor mínimo de colisiones encontrado.

```
results_table <-
  data.frame(tamaño_poblacion = pop_sizes, num_min_choques = collisions)
print(results_table)
```

```
##   tamaño_poblacion num_min_choques
## 1               50               6
## 2              100               8
## 3              200               7
## 4              500               9
## 5             1000              10
## 6             2000               9
```