LAB 5:

Traveling Salesman Problem using Genetic Algorithms

COMPUTATION

Spring semester. Academic year 2023/2024

Eduardo Ezponda Igea
Ingeniería Informática Internacional
UPNA

INDEX:

PROBLEM DEFINITION:

For the final project, a genetic algorithm has been implemented to solve the salesman problem. The problem tries to find out what route a traveler must take to visit all the places from beginning to end, without being able to visit any again, and returning to the starting point.

These types of problems are characteristic because it is not known with certainty when the optimal solution is reached. Yes, you can know when you have obtained a good solution, but it is difficult to know if it is the best solution to the problem.

For the case of the problem presented, we have a set of 20 Spanish cities, with the distance one to one from all the cities. In our case, we will start from Pamplona and, therefore, we will end in Pamplona.

CHROMOSOME REPRESENTATION:

At the beginning of the project, it was decided to assign Pamplona in the first and last position of a table of 21 positions. However, the salesman's problem is a circuit. This means that we would be losing 1 position to obtain better results since only 19 positions are modified. Therefore, the representation of the individuals was changed to a table of 20 positions, and at the end of the algorithm, the route would be obtained starting in the city of origin (Pamplona). In this way, the number of permutations would be greater than in the previous case.

Since the representation of the individual has the total set of cities and their respective order, the cost of the function can be obtained. This cost will be calculated simply by adding, from the first position to the penultimate position, the distances between the current city and the city of the next position. At the end of that loop, keep in mind that you have to add the distance from the last city in the table to the first position. In

this way, it will be obtained which chromosomes have a shorter path, and therefore, are the best solution.

In this case, the fitness function is the same as the cost function because it has been taken into account that a better solution is the one with the lowest fitness. However, it would be normal for the best solutions to be those with the highest fitness, and therefore, the cost would be the inverse of the fitness.

All this will be carried out taking into account the matrix M provided. This matrix has the distances of all the cities, allowing the cost of traveling through all of them to be calculated. The index of the matrix will be taken into account in the table directly, and not the name of the city, to make the code more efficient.

```
def fitness(population):
    fitness_scores = np.zeros(len(population))

    for i in range(len(population)):
      for j in range(NUMBER_OF_CITIES - 1):
        fitness_scores[i] += M[population[i][j]] [population[i][j+1]]
      fitness_scores[i] += M[population[i][0]] [population[i][NUMBER_OF_CITIES - 1]]

    return fitness_scores
```

As can be seen in the image, the fitness of each of the individuals in the population is calculated. To do this, you go through each of the cities in the second for, adding the distance to the next position. When it is finished, the distance from the last position to the first is added.

OBJECTIVES:

In this practice, several factors will be taken into account. Those factors are whether a solution is reached, the time cost, and the number of iterations. Various tests will be carried out showing results with each of the methods. In this way, an analysis will be obtained to specify the reason for each solution and which genetic algorithm obtains the best results.

PARENT SELECTION:

As methods for assigning probabilities to an individual as a parent, the Roulette method and the Tournament method with replacement and without replacement have been implemented.

In the Roulette method, chromosomes are selected with a probability proportional to their fitness. This means that chromosomes with better/lower fitness have a higher probability of being selected. This method has a convergence problem because individuals with better fitness tend to quickly dominate the population. To achieve a better solution, a possible solution has been implemented with Goldberg Sigma Scaling. Thanks to this, we do not use the original fitness function but rather an adaptation taking into account the mean and the deviation from the mean.

Secondly, you have the Tournament method. In this method, a fixed number of chromosomes (k) is selected randomly, and the best chromosome in the group is chosen as the parent. This process is repeated to select all necessary parents.

This k value is difficult to choose since depending on the value you choose, you will have different results. Generally, a smaller k value results in less selection pressure because with fewer individuals in each tournament, less fit individuals have a higher probability of being selected. However, this produces greater genetic diversity, and in the

long run of iterations, an aid to not converge at the beginning of the algorithm.

In the opposite case, when a larger k is chosen, the tournaments are more competitive as there are more individuals in each tournament. This implies that fitter individuals have a higher probability of being selected. However, it produces less genetic diversity which can lead to the most optimal individuals quickly dominating the population and, consequently, obtaining faster convergence of the algorithm.

In this case, a k value of 5 has been chosen. In this way, greater selection pressure will be obtained but maintaining a little of the genetic diversity of the population.

The same thing happens with the choice of the Tournament method with or without replacement. If chosen with replacement, the same individual can be in the same tournament more than once. On the contrary, without replacement implies that that individual can only be in the tournament once. Both methods will be tested to determine if the method with replacement produces a convergence problem for the fittest individuals.

CROSSOVER:

In our problem, each individual is represented as a set of values ranging from 0 to 19, inclusive. Each index represents a city. Therefore, the chromosome presented is a permutation chromosome. This leads us to have chosen for the crossover method part, the Partially Mapped Crossover (PMX) and Order Based Crossover methods.

In the case of binary or floating point chromosomes, this implementation would have been much simpler because the repetition of the values that the chromosome takes would not have to be taken into account. In the case presented, it must be ensured that any individual in the population has all the different values, and that each of them is in the range from 0 to 19.

The main difference between the two chosen crossover methods is that PMX ensures that offspring inherit segments from their parents while maintaining a valid permutation. However, the Order Based Crossover method focuses on preserving the relative order of the genes (cities) of the parents in the offspring.

The probability that has been assigned to the crossing methods is 0.95 or 95%. In the small case in which there is no cross, the two parents will remain as if they were the two descendants of the cross.

SURVIVOR SELECTION:

In the selection section of the survivor selection method, the fitness-based replacement method has been chosen. This means that the individuals from the population with the best/highest fitness will be the survivors that persist in the new population.

Furthermore, the population size remains constant throughout the problem. Therefore, if we have a number of parents that is 100, another 100 descendants are obtained. Of that total set of 200 individuals, 100 of them will be rejected based on their aptitude.

MUTATION:

For the mutation, the presence or absence of the method will be taken into account. Typically, mutation can have a very significant impact on the performance of the algorithm and the quality of the solutions found. Although it may make the first solutions worse at first, in the long run it produces a lower probability of premature convergence because it introduces random variations in the chromosomes. This involves an exploration of a larger space of solutions not reached solely by the crossover method.

Furthermore, a mutation cannot always produce an individual with lower aptitude, but it can even improve it, obtaining greater genetic variability.

The probability of mutation will be 0.1 or 10%. This probability implies that a bad result is not obtained in the long term by choosing a very high mutation, and therefore, that the fittest individuals are not taken into account in the new populations. That 10% will mean that there is no great risk of premature convergence.

In this case, all possible mutations found in the theory have been implemented. These mutations are the following:

1. Exchange mutation:
    a. It consists of exchanging the position of two elements in the sequence. Thanks to this, new configurations can be explored quickly, improving genetic diversity.

2. Insertion mutation
    a. An element of the sequence is extracted and reinserted at a new position. This facilitates the final adaptation of the solution by allowing more precise adjustments.

3. Shake mutation
    a. Make multiple small perturbations in the sequence. This causes premature convergence to be avoided by exploring the solution space more exhaustively.

4. Inversion mutation
    a. Reverses the order of a subsection of the sequence. Therefore, it promotes the search for optimal solutions by reorganizing large segments of data efficiently.

As mentioned at the beginning of the report, the initial city will be Pamplona. However, you can change the city to any other. If the first

case had been taken into account in which the initial city was fixed in the first and last gene in the chromosome representation, changing the initial city could lead to a different solution being obtained.

Having implemented the second solution in which the individual is a circuit, the final result is obtained when the number of iterations ends. Therefore, the solution will be the same choosing any city as the starting city.

Finally, before starting the tests, the number of cities in the individuals will be varied to draw possible conclusions regarding changes in the solution obtained.

TESTING:

The following tests will be carried out in different executions of the program. To do this, we have separated it into a function called genetic Algorithm in which the parent selection method, the crossover method, and finally, the mutation method are passed as parameters.

The population will be 100 individuals, and the number of parents will have the same size as the population (100). The number of iterations will also be 100.

As mentioned above, the variable k will have the value of 5 for the Tournament method.

A number of 3 executions will be carried out for each test to vary the population and thus have a more complete solution. Otherwise, a single population could lead to solutions unforeseen by that specific population.

In addition, the time it took to execute the algorithm, the final fitness results obtained, and a graph of the evolution of the lowest fitness per iteration will be shown.

Firstly, the algorithm will be executed 3 times for each of the parent selection methods that are implemented (Roulette, Tournament with replacement, Tournament without replacement). The Exchange method has been chosen for the mutation, and PMX for the crossover.

The results are the following:

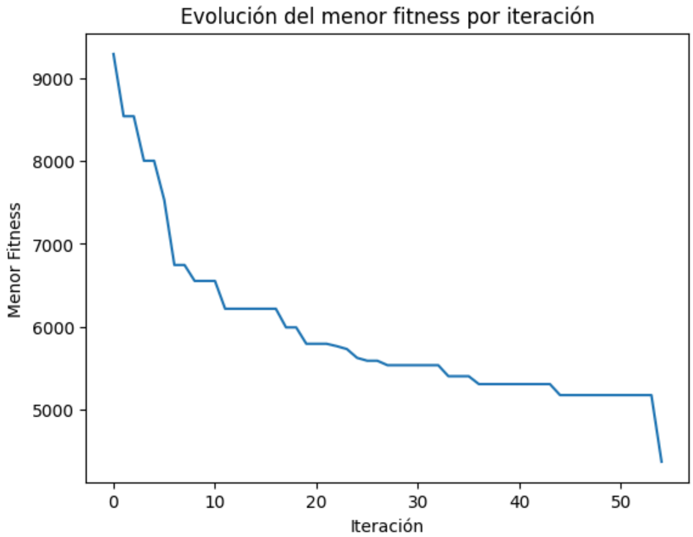
METHODS:

- Crossover method: Partially Mapped Crossover
- Mutation method: Exchange
- Survivor selection method: Fitness-Based Replacement

CONSTANTS:

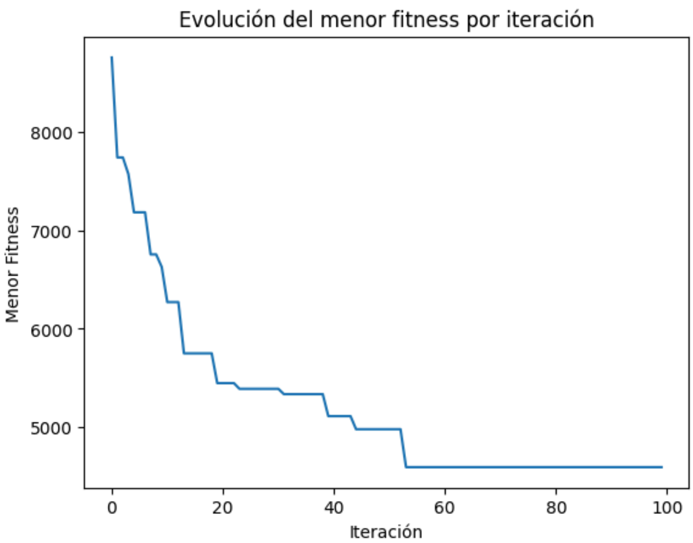
- Number of iterations: 100
- Probability of crossover: 0.95
- Probability of mutation: 0.1
- Population size: 100
- Number of parents: 100
- Position origin: Pamplona

```
Solución de la iteración 1: [11, 0, 18, 16, 7, 1, 13, 14, 2, 12, 6, 9, 19, 10, 17, 3, 8, 4, 15, 5], Fitness: 4813.0
Parent selection method:  Roulette
Fitness  [4367. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171.
 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171.
 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171.
 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171.
 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171.
 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171.
 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171.
 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171. 5171.
 5171. 5171. 5171. 5171.]
```



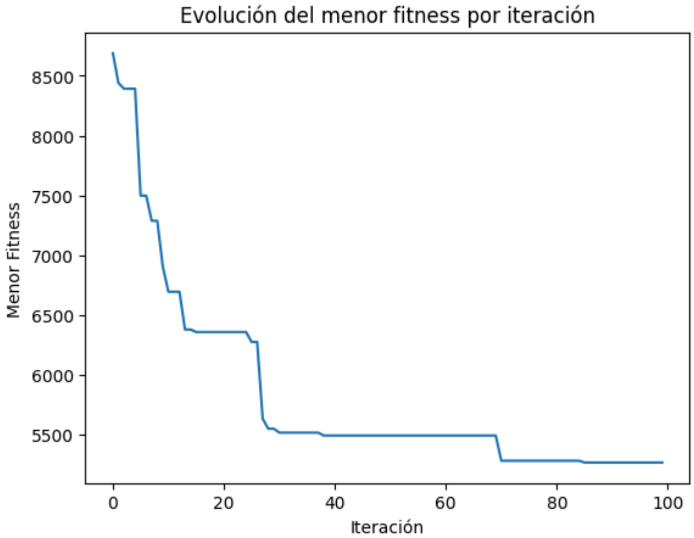

```
Solución de la iteración 2: [5, 4, 15, 8, 3, 17, 10, 9, 6, 12, 2, 14, 13, 19, 1, 7, 16, 18, 0, 11], Fitness: 4367.0
```

```
Parent selection method:  Tournament with replacement
Fitness  [4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595.
 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595.
 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595.
 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595.
 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595.
 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595.
 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595.
 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595. 4595.
 4595. 4595. 4595. 4595.]
```




```
Solución de la iteración 3: [18, 0, 11, 17, 5, 15, 4, 8, 3, 10, 9, 6, 12, 2, 14, 13, 19, 1, 7, 16], Fitness: 4595.0
```

```
Parent selection method:  Tournament without replacement
Fitness  [5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264.
 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264.
 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264.
 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264.
 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264.
 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264.
 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264.
 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264. 5264.
 5264. 5264. 5264. 5264.]
```




```
Solución de la iteración 2: [13, 2, 14, 7, 1, 16, 0, 11, 5, 4, 15, 8, 3, 6, 12, 9, 10, 17, 18, 19], Fitness: 5264.0
```

As can be seen in the images, the iteration of each parent selection method that has obtained the best result has been chosen. In general, quite similar results have been obtained but slightly superior depending on the initial population. This means that it will not be a problem to take into account the best solution instead of the average solution for the algorithm analysis.

The best solution for the algorithm is 4367 kilometers. In this case, quite satisfactory results have been obtained that are close to the optimal solution, and are even achieved. Furthermore, the Roulette method is in this case the one that obtains a better solution.

It can also be seen in the graphs how the Roulette method is the one that takes the longest to converge, and even does not do so. This tells us that in the long term, if the number of iterations were increased, it is very likely that the optimal solution would be reached, without taking into

account the unforeseen event of having reached the solution with an optimal case of mutation.

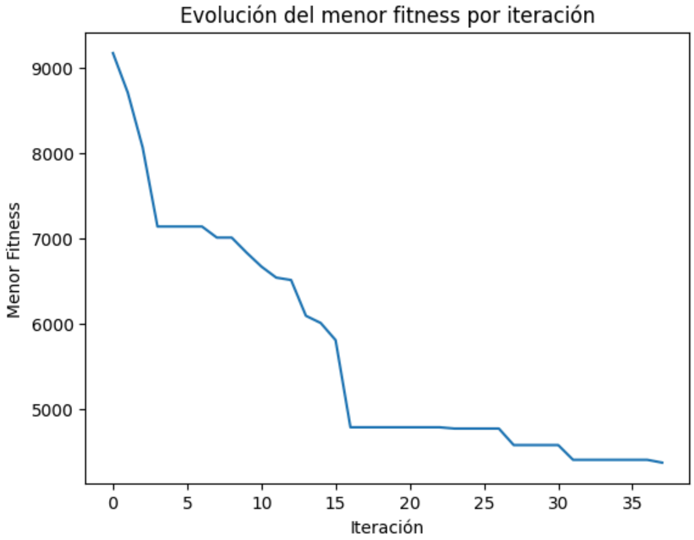
When the Tournament with replacement method is chosen, it implies that the fittest individuals have a greater probability of dominance in the population. This means that better solutions can be obtained, but with a greater risk of convergence of the algorithm.

As shown in the second graph, it is the first of the 3 methods to converge. In the first part of the algorithm, up to iteration 20, it progresses very quickly. However, you then get pretty stuck, but slowly progressing until you probably get a successful mutation at iteration 50 that manages to lower the fitness to get a better solution.

Finally, we have the Tournament without replacement that has 3 convergences throughout the algorithm, but thanks to the 0.1 probability of mutation, it manages to progress when a more suitable individual is found. It is also advisable in this case to increase the probability of mutation or decrease the value of k to provide greater genetic diversity. By testing the previous variations, the solutions have been reduced by several hundred kilometers.

With the previous changes, it has even been possible to obtain the optimal solution of the algorithm. To do this, the probability of mutation has been increased to 0.15 and the variable k has been given a value of 3.

```
Parent selection method:  Tournament without replacement
Fitness  [4367. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400.
 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400.
 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400.
 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400.
 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400.
 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400.
 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400.
 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400. 4400.
 4400. 4400. 4400. 4400.]
```




```
Solución de la iteración 2: [18, 16, 7, 1, 19, 13, 14, 2, 12, 6, 9, 10, 17, 3, 8, 15, 4, 5, 11, 0], Fitness: 4367.0
```

As a final result of the first tests, the best solution has been obtained with the Roulette method. The execution time of the 3 iterations with each of the methods has been 38 seconds, obtaining an average of (38/9) 4 seconds per genetic algorithm executed. It is true that it may be too long, but the code is not optimized in terms of efficiency and lists.

The best solution found makes the traveler travel through the cities in the following order:

Pamplona → Zaragoza → Barcelona → Girona → Tarragona → Valencia → Alicante → Murcia→ Córdoba → Cádiz → Sevilla → Huelva → Cáceres → Toledo → Madrid → León → Coruña → Oviedo → Bilbao → Donostia→ **Pamplona**

It is not only the best solution found, but the best solution of the algorithm.

```
Mejor solución global encontrada:
['Córdoba', 'Cádiz', 'Sevilla', 'Huelva', 'Cáceres', '
Fitness global: 4367.0
Tiempo de ejecución del algoritmo: 38.04294180870056
```

TEST 2:

Next, another test will be carried out modifying the methods that will be implemented. In this case, we are going to change the crossover method to use the Order Based method. Furthermore, we will remove the mutations so that the elements obtained in the crossing will be directly evaluated for their possible insertion into the new population.

Finally, the last 3 cities are going to be eliminated, which are Toledo, Valencia and Zaragoza. The city of origin has been changed to Madrid.

METHODS:

- Crossover method: Order Based Crossover
- Mutation method: No mutation or Shake mutation
- Survivor selection method: Fitness-Based Replacement

CONSTANTS:

- Number of iterations: 100
- Probability of crossover: 0.95
- Probability of mutation: null or 0.1
- Population size: 100
- Number of parents: 100
- Position origin: Madrid

In the following image, you can see how the function line of the PMX method and the execution of the exchange mutation have been commented on.

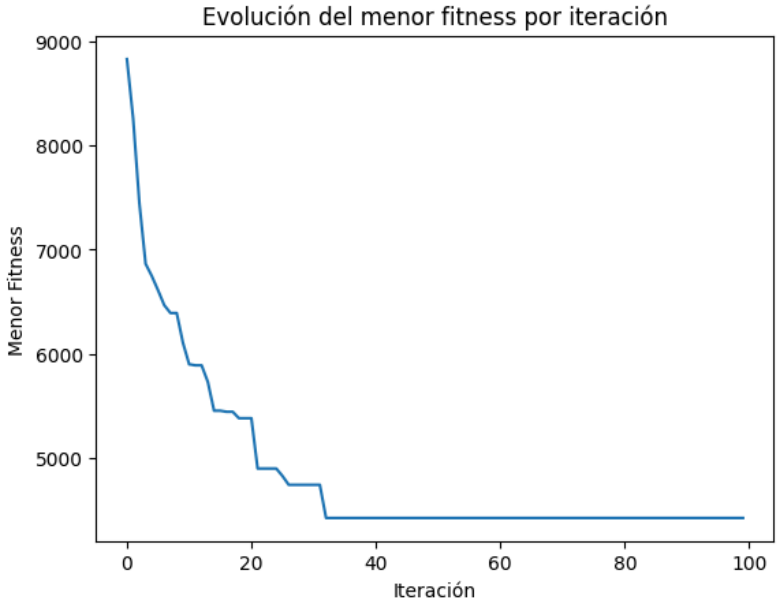
```
# Crossover and mutation
new_population = []
for i in range(0, NUMBER_OF_PARENTS, 2):
  if i + 1 < NUMBER_OF_PARENTS:
      #PMX
      #descendant1, descendant2 = partiallyMappedCrossOver(parents[i], parents[i+1], prob_cross)
      #ORDER-BASSED
      descendant1, descendant2 = orderBassedCrossOver(parents[i], parents[i+1], prob_cross)

      #new_population.append(exchangeMutation(descendant1, prob_mut))
      #new_population.append(exchangeMutation(descendant2, prob_mut))
      #NO MUTATION:
      new_population.append(descendant1)
      new_population.append(descendant2)
```

In this piece of code you can see how it has been modified to eliminate the last 3 cities and thus reduce the total set, and how Madrid has been placed as the city of origin.

```
2 NUMBER_OF_CITIES = len(CITIES) - 3
3 POSITION_ORIGIN_DESTINATION = 10
```

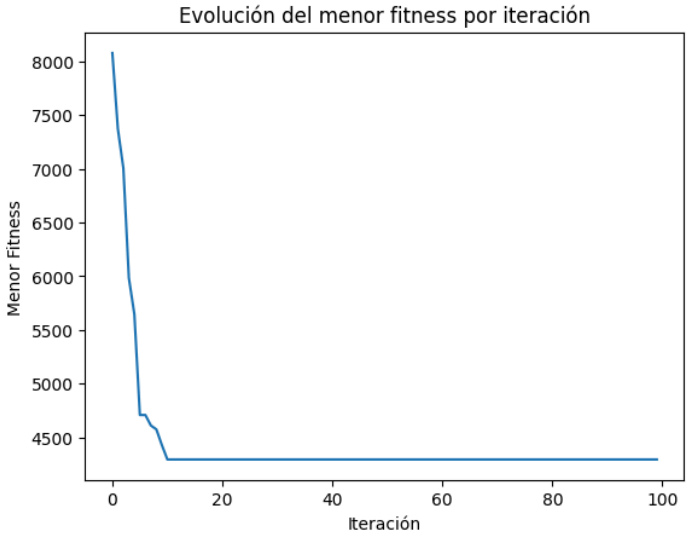
The results obtained are the following:

```
Parent selection method:  Roulette
Fitness  [4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429.
 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429.
 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429.
 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429.
 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429.
 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429. 4429.
 4429. 4429. 4429. 4429.]
```



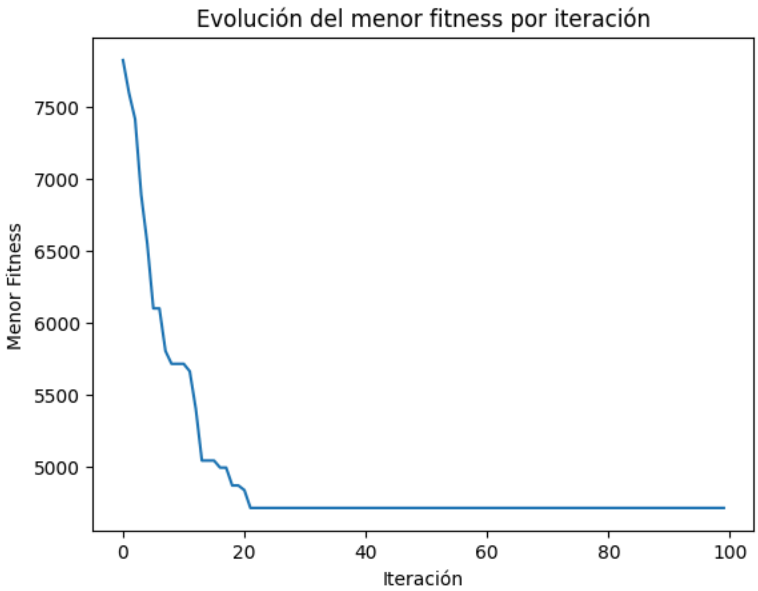

```
Solución de la iteración 2: [14, 2, 12, 6, 9, 10, 3, 5, 8, 15, 4, 11, 0, 16, 7, 1, 13], Fitness: 4429.0
```

```
Parent selection method:  Tournament with replacement
Fitness  [4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295.
 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295.
 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295.
 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295.
 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295.
 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295.
 4295. 4295. 4295. 4295.]
```




```
Solución de la iteración 1: [7, 1, 16, 0, 11, 5, 15, 4, 8, 3, 10, 9, 6, 12, 2, 14, 13], Fitness: 4295.0
```

```
Parent selection method:  Tournament without replacement
Fitness  [4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716.
 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716.
 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716.
 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716.
 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716.
 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716.
 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716.
 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716. 4716.
 4716. 4716. 4716. 4716.]
```




```
Solución de la iteración 3: [15, 8, 3, 10, 13, 14, 2, 12, 6, 9, 1, 7, 16, 0, 11, 5, 4], Fitness: 4716.6
```

As can be seen, smaller fitness values are obtained. This is explained because by reducing the number of cities, the total sum of kilometers to travel through the cities is lower.

In this case, quite satisfactory results have been obtained also without applying the mutation. However, you can see that the convergence is much higher because once it converges, there is no way to get better results.

Despite that, and in addition to the Tournament with replacement method having a great risk in being able to choose the same individual in the same tournament, it is the method that has obtained the best results.

However, it can be seen that of the 100 iterations provided for the genetic algorithm, 85 iterations are discarded because from iteration 15 onwards the algorithm converges. In the same way but a few iterations later, the Tournament without replacement method converges at iteration 20, and the Roulette method at iteration 30.

Therefore, it can be concluded that making mutations to individuals in the algorithm is essential to guarantee a better solution in the long term, although in the short term it may progress more slowly.

We are going to implement this same algorithm but adding the shake mutation method. This method randomly alters a part of the chromosome. Unlike the exchange mutation, in the short term this method can obtain much less fit individuals because the exchange mutation simply varies the value of two genes.

In this case, the best fitness value is 4295, and it has been obtained in 6 of the 9 total executions. This emphasizes the importance of the mutation method to obtain better results in the long term, and thus converge later. Since it has not been checked with the value 4295 and was previously checked with 4367, the algorithm is left performing unnecessary iterations.

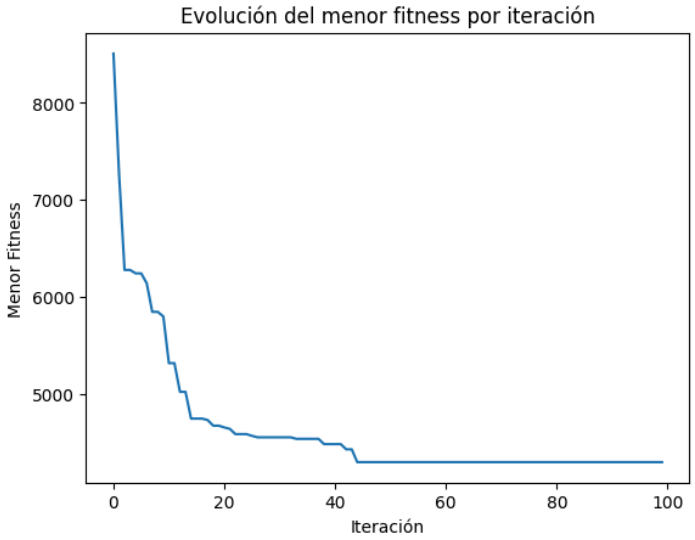
The traveler's route to travel the shortest distance having eliminated the last 3 cities is as follows:

Madrid → León → Coruña → Oviedo → Bilbao → Donostia → Pamplona → Girona → Barcelona → Tarragona → Alicante → Murcia → Córdoba → Cádiz → Sevilla → Huelva → Cáceres → **Madrid**

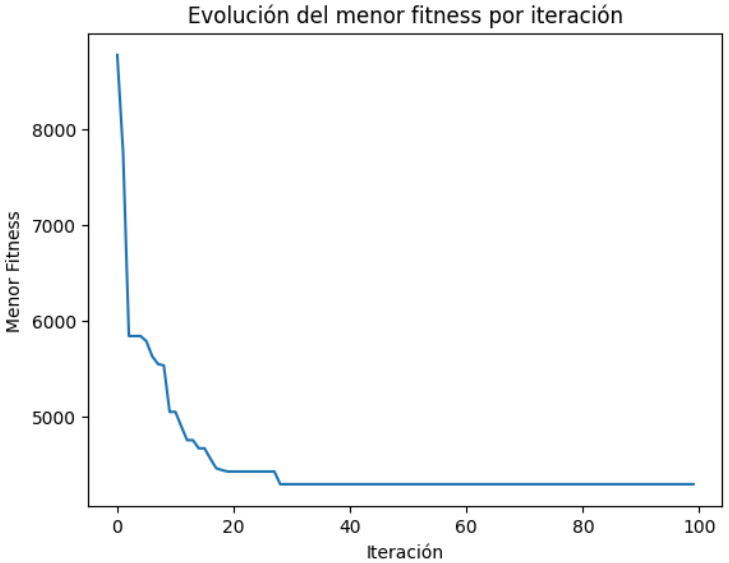
The execution time of the algorithm was 37.76 seconds, taking an average of 4 seconds per execution (very similar to the other tests but in this case having fewer cities).

The results obtained are the following:

```
Parent selection method:  Roulette
Fitness  [4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295.
 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295.
 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295.
 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295.
 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295.
 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295.
 4295. 4295. 4295. 4295.]
```

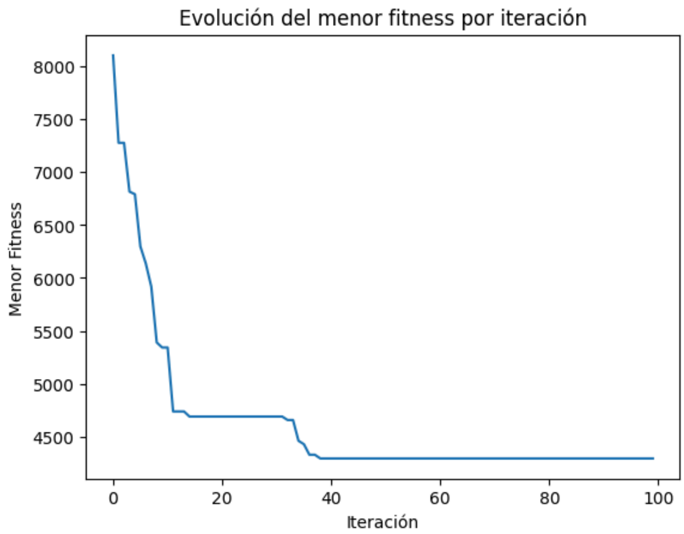



```
Solución de la iteración 2: [16, 7, 1, 13, 14, 2, 12, 6, 9, 10, 3, 8, 15, 4, 5, 11, 0], Fitness: 4295.
Parent selection method:  Tournament with replacement
Fitness  [4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295.
 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295.
 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295.
 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295.
 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295.
 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295.
 4295. 4295. 4295. 4295.]
```




```
Solución de la iteración 1: [4, 15, 8, 3, 10, 9, 6, 12, 2, 14, 13, 7, 1, 16, 0, 11, 5], Fitness: 4295.
```

```
Parent selection method:  Tournament without replacement
Fitness  [4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295.
 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295.
 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295.
 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295.
 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295.
 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295.
 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295.
 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295. 4295.
 4295. 4295. 4295. 4295.]
```




```
Solución de la iteración 2: [4, 5, 11, 0, 16, 7, 1, 13, 14, 2, 12, 6, 9, 10, 3, 8, 15], Fitness: 4295.0
```

It should be noted that in the 3 executions of the Roulette method, the best solution has been obtained. In turn, 2 of the 3 executions of the Tournament with replacement method have also obtained the optimal solution.

The Roulette method remains the last to converge as it has more varied parents, and not always the most suitable.

The Tournament with replacement method converges quickly, so it is necessary to increase the mutation probability to a value greater than 0.1. A possible value could be 0.2.

The crossing method does not vary much between using PMX or Order Based, because similar values are still obtained.

Changing the city of origin does not change, since as explained previously, the problem is a circuit that starts from the final solution in which the route is obtained from the final result.

Finally, adding or removing random cities will generally not cause problems in the result. The relationship will be similar to directly proportional when you have many cities, unless cities are added or removed on purpose that cause a much longer journey. In tables with few cities, eliminating or adding cities can cause very large changes in the results of the sum of kilometers to be traveled.

CONCLUSION:

In conclusion, it can be highlighted that in general, the results obtained by a genetic algorithm to solve the salesman problem are very satisfactory. Solutions close to the optimal solution are usually obtained.

A variable to take into account is the size of the population. It has been observed that when the population size is reduced to small values, the solutions obtained are poor.

The opposite happens with the number of iterations, which, as can be seen in the graphs, could have reduced the number of iterations by several dozen. It is true that when using the Roulette method it is not the most optimal, but in the Tournament methods very similar values would be obtained with a much lower computational cost.

Finally, we want to highlight the importance of the mutation method to extend the convergence of the algorithm over time. Thanks to this, much more optimal results are obtained.