Optimizing the Travelling Salesman Problem with genetic algorithms:

Spain cities version

Nombre: Guillermo Azcona Recari

NIA: 141107

Fecha: 14 de mayo de 2024

Universidad Pública de Navarra

Index

Definition of the problem

Initialization of Chromosomes

Evaluation of Chromosomes (Fitness)

Selection of Parent Evaluation Methods

Crossover Functions for Chromosomes

Mutation Methods

The Algorithm

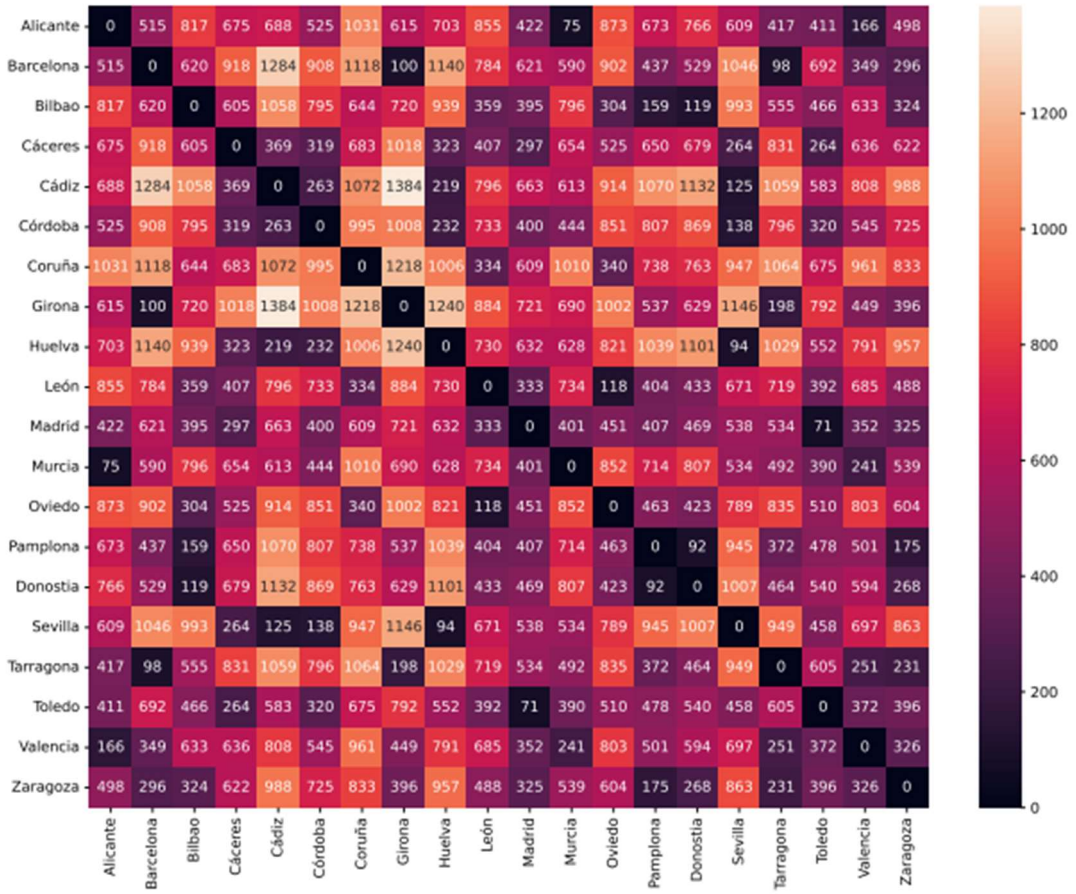
Algorithm configuration

Conclusion

Definition of the problem

This practice revolves around the exploration of **genetic algorithms** as a solution method for a defined problem: determining the most cost-effective route to visit each of **20 specified Spanish cities starting and ending in Pamplona**, ensuring each city is visited exactly once.

For this project we will use a cost matrix as the basis for calculating travel costs between cities, adhering to the constraint that travel costs are symmetrical.



This practice tries to solve the **Travelling Salesman Problem**, a classic problem in operations research and computer science, aiming to find the shortest possible route that allows a salesman to visit each city exactly once and return to the original city, completing a loop that starts in Pamplona and finishes in this same city.

Initialization of Choromosomes

The chromosomes are composed of 20 positions in which every position represents a gen. Every gen corresponds to the index position of a city and are not repeated (permutation).

Among the various options considered for this project, we stood two.

The first involved fixing Pamplona at the first position of the chromosome array and randomly generating the remaining sequence, thereby **creating random permutations of 19 positions**. Here is an example (where Pamplona is index 13):

[13, 4, 3, 19, 6, 14, 2, 1, 18, 15, 12, 5, 11, 7, 0, 16, 9, 8, 10, 17]

The second option, involves generating **completely random permutations for all 20 positions**, effectively treating the route as a circuit where each position, including Pamplona, could appear anywhere in the sequence.

Generating chromosomes as in this example below:

[4, 3, 19, 6, 14, 2, 1, 18, 15, 12, 5, 11, 7, 0, 16, 13, 9, 8, 10, 17]

Could give us some advantages versus the first option such as **increase the diversity in the initial population as** we allow any city to appear at any position and explores a wider variety of potential solutions from the start and escaping from evaluate only the local optima. In addition, **all the cities have the same treatment because**, as not all the routes start from Pamplona, we can discover novel routes that might be overlooked when Pamplona is always the starting point.

Evaluation of Chromosomes (Fitness)

After creating the chromosomes, we should evaluate his accuracy, to make this we designed a function that sums all the distances from the first position to the last and the one from the last to the first one to ensure the route forms a **complete loop**, thus accurately representing a potential solution for the Travelling Salesman Problem.

Here we find a visual representation of how it works:

[4, 3, 19, 6, 14, 2, 1, 18, 15, 12, 5, 11, 7, 0, 16, 13, 9, 8, 10, 17]

As we can see, a better solution will be a solution with a lower fitness as it represents the kilometres travelled.

Selection of Parent Evaluation Methods

I have implemented two distinct parent selection methods, **tournament and roulette** method, that plays a crucial role maintaining genetic diversity and ensuring the convergence towards optimal or near-optimal solutions.

Firstly, the **TOURNAMENT METHOD** involves organizing a mini-competition or "tournament" among a subset of the population.

In this method, **k random** chromosomes are selected from the population, with the extent of k allowing the algorithm to control the selection pressure.

A smaller k value leads to lower selection pressure, whereas a larger k promotes the selection of stronger candidates.

The **replacement** parameter dictates if the same individual can be selected multiple times for different tournaments, improving the diversity of the resultants.

I implemented also a **p** parameter, that introduces a probabilistic element in selecting the tournament winner.

If p = 1.0 always selects the best individual from the tournament, while a lower p value increases the chances for less fit individuals to be selected and preserves genetic diversity.

Secondly, the **ROULETTE METHOD** makes that the chromosome's probability of being selected is proportional to its fitness. As the objective is to minimize route costs, the function uses the inverse of the fitness values.

Chromosomes with lower route costs have a higher probability of selection. The method involves calculating the cumulative probability distribution from these normalized.

Inverted fitness values and selecting parents based on these probabilities.

This stochastic approach favors those who are more fit and favourises the survival of the fittest.

Crossover Functions for Chromosomes

To mix genetic information from two parents to create effective new chromosomes, I used 3 different crossover methods.

Firstly, a **variation of the CROSSOVER ON A POINT**, where in this method splits each parent's route at a random point and merges the first part of one parent with the remaining unique cities from the other.

I just implemented to try if a integer and binary method could give me a good result as this crossover favourises the introduction of very different combinations.

Secondly, the **PMX (Partially Mapped Crossover)** method, in which a segment from each parent is directly copied into the offspring. The rest of the route is filled by mapping the remaining cities to avoid duplicates, respecting the original sequence.

This method can give to the algorithm a good keeping of the ordering of the cities and specially being effective in cases where the order of the routes impacts the overall travel cost, preserving the better sequences of cities.

Thirdly, the **ORDER-BASED** method that copies a sequence from each parent into the offspring and fills the gaps with cities from the other parent in the order they appear being sure it not overlaps.

Mutation Methods

The mutation methods are implemented to create, in every case and with a determinate probability a increase on the population diversity, exploring new routes and solutions.
I have implemented four different methods.

Firstly, the **SWAP MUTATION**, which swaps two cities in the chromosome if a random threshold is lower than the mutation rate, introducing minor changes in the route that might lead to improvements in the total travel distance.

Secondly, the **INSERTION MUTATION** which removes a city from the chromosome and inserts it at a different position. This mutation alters the order of the cities, potentially discovering more efficient paths.

Thirdly, the **SHAKE MUTATION** which randomly selects two cities and swaps them, similar to the swap mutation but more focused on creating variability across the route.

Las but not least, the **INVERSION METHOD,** which selects a segment of the chromosome and reverses the order of cities within that segment. This can significantly rearrange the route.

After mutation, I apply the survivor method selected for this project, the **GENITOR**. This method replaces the least fit individuals in the population with the best of the offspring from the current generation. The adoption of the GENITOR approach offers come advantages such as finding a fast convergence by eliminating the worst chromosomes, and could be verty useful in early stages, to approach the solution faster. For this reason, I chose this one for the project, which combines finding a the best solution in the less time possible.

The Algorithm

Finally, the algorithm is included in a function called genetic_algorithm and, after initialization the random population, it evolves the population through selection, crossover, and mutation processes, aiming to find the shortest possible route.

It iterates and finds solutions over multiple generations and visualizing the improvement in route efficiency to last return the best solution found.

We call this function from a piece of code which aim is to compare between the same genetic algorithm configuration and **obtain an objective view of the results**.

Algorithm configurations

Now, I will evaluate the possibilities of the algorithm, changing its values and methods to obtain the best solution.

We first want to try the different parent selection method; we will first implement a basic configuration with the tournament selection method.

1st Case

Parent selection method: `Tournament Selection (k=3)`

Crossover method: `PMX`

Mutation method: `Swap Mutation (mutation_rate=0.05)`

Survivor method: `Genitor`

Iterations: `100`

Population Size: `50`

Times executed the algorithm: `100`

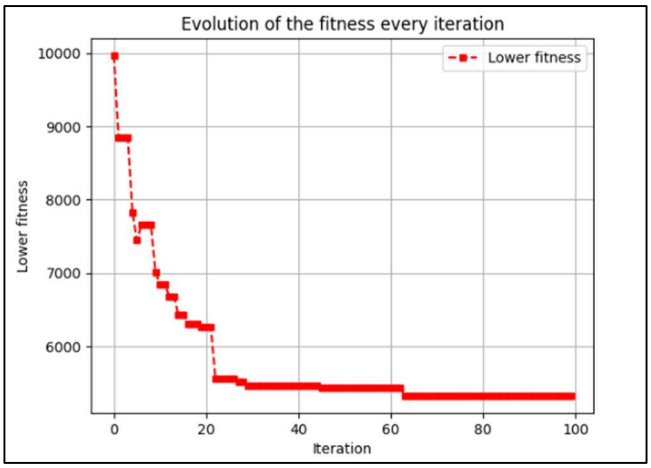
In this case, we used a small k value to maintain diversity by reducing selection pressure, preventing premature convergence. Then, we used PMX, that as I said before, respects the order of cities generally, which is crucial for TSP where the sequence affects the total distance, and the swap mutation explores nearby solutions without drastically altering successful routes. Also, the number of iterations is enough to allow the population to evolve and stabilize in the performance and we with a moderate population size to provide enough genetic variation without a very exigent computation.

Results 1st Case

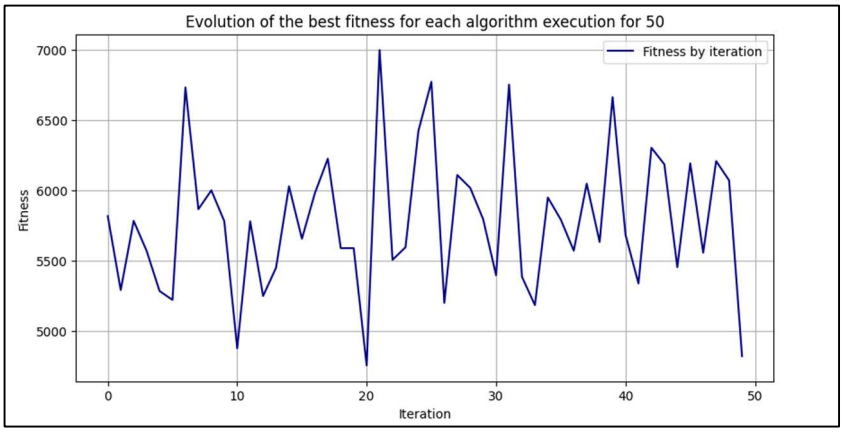
```
Time of execution for a random execution of the algorithm
0.596226692199707

Number of fitnesses calculated for an execution of the algorithm
5000
```

Graph [1.1]: Evolution of the fitness every iteration for a random execution



Graph [1.2]: Evolution of the best fitness for each algorithm execution for 50 times



Interpretation of results:

As in one random execution, the best solution was:

['Oviedo', 'Bilbao', 'Zaragoza', 'Alicante', 'Murcia', 'Córdoba', 'Cádiz', 'Sevilla', 'Huelva', 'Cáceres', 'Toledo', 'Madrid', 'Valencia', 'Tarragona', 'Girona', 'Barcelona', 'Pamplona', 'Donostia', 'Coruña', 'León'].

with fitness of 5430, we say the best route starts from Pamplona, continues to Donostia or Barcelona and then returns from the other side as is a circuit. After trying more times, I realize was an irregular configuration, far from the solution, and as it's shown in the Graph 1.2, after executing it for 50 times, we can see it's best result is changeable between 4754 and 7000. We can deduce one problem why this case doesn't reach a good solution is because k=3 could end up reducing diversity too quickly and as we can see in Graph 1.1, there is a early stabilization of the best fitness. Also, as we use of PMX crossover and a low mutation rate, this might not introduce enough new genetic material to escape local optima.

As we can blame on the diversity as one of the factor of irregularity and stabilization, we are going to create a **hight diversity case, using the roulette method**.

2nd Case

Parent selection method: `Roulette Wheel Selection`

Crossover method: `Order-Based Crossover`

Mutation method: `Inversion Mutation (mutation_rate=0.1)`

Survivor method: `Genitor`

Iterations: `200 Generations`

Population Size: `100`

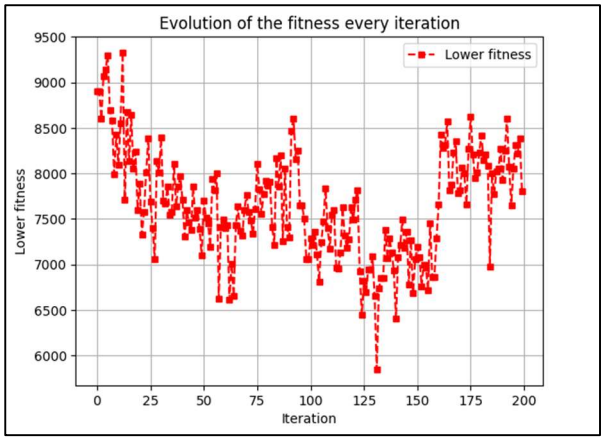
Times executed the algorithm: `50`

In this case, we use the Roulette method as it offers every individual a chance to be selected and maximizes the genetic diversity. In addition we add the order-based crossover, that preserves sequences of cities effectively and can discover new configurations that in PMX or others is not easy to do. Moreover, I increased the mutation probability and the iterations to also explore a more varied population.

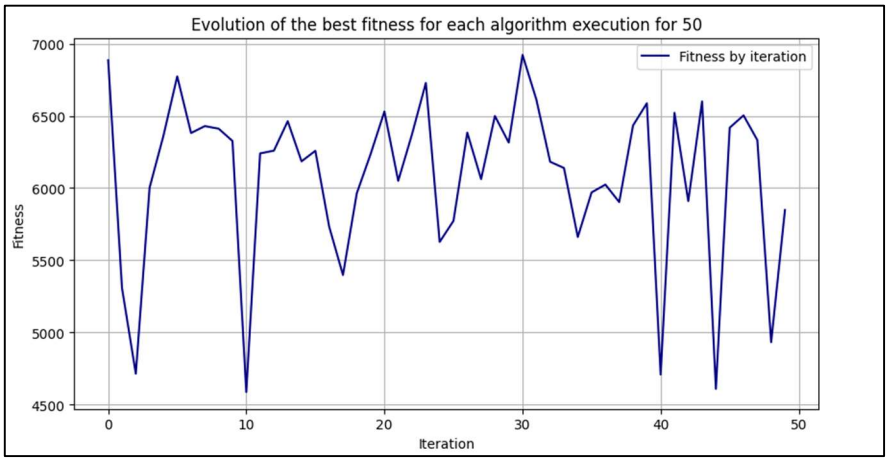
Results 2nd Case

```
Time of execution for a random execution of the algorithm:
                    1.131329298019
  Number of fitnesses calculated for an execution of the
                      algorithm:
                        40000
```

Graph [1.1]: Evolution of the fitness every iteration for a random execution



Graph [1.2]: Evolution of the best fitness for each algorithm execution for 50 times



Interpretation of results:

As in one random execution, the best solution was:

```
['Madrid', 'Toledo', 'Pamplona', 'Zaragoza', 'Tarragona', 'Barcelona',
     'Girona', 'Donostia', 'Bilbao', 'Coruña', 'León', 'Oviedo',
  'Alicante', 'Murcia', 'Valencia', 'Córdoba', 'Sevilla', 'Huelva',
                        'Cádiz', 'Cáceres']
```

with fitness of 5848, we say the best route starts from Pamplona, continues to Zaragoza or Toledo and then returns from the other side as is a circuit. As we can see in Graph 2.1, we experiment a slower convergence because as a larger population size and the use of roulette wheel selection promote diversity, it also slow down the convergence rate, requiring more generations to find an optimal solution. In addition, we evaluate the complexity of computation, where the time is much higher (almost the double) and the number of calculations of the fitness are eight times higher. Knowing this, seems the results are better not because of the diversity of

generation principally (affection negatively to the convergence), but because of the number of generations.

Knowing this, I will prioritize in the 3rd case the convergence and I will try to do it fast.

3rd Case

Parent selection method: `Tournament Selection (k=5)`

Crossover method: `Crossover on a point`

Mutation method: `Swap Mutation (mutation_rate=0.01)`

Survivor method: `Genitor`

Iterations: `50`

Population Size: `30 chromosomes`

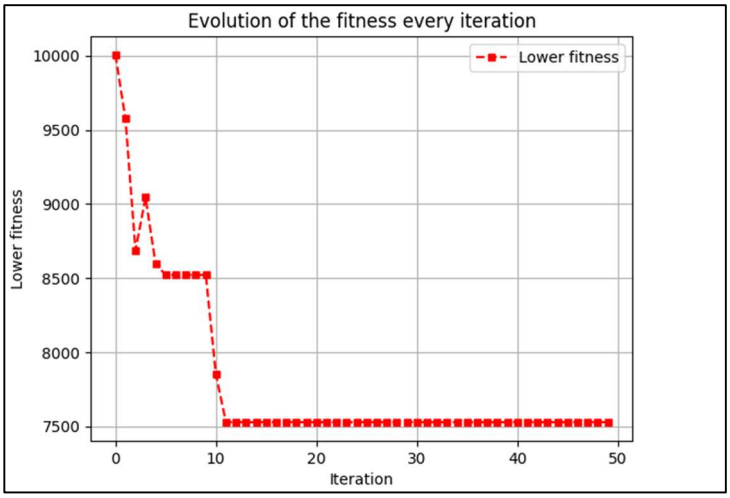
Times executed the algorithm: `50`

In this case,

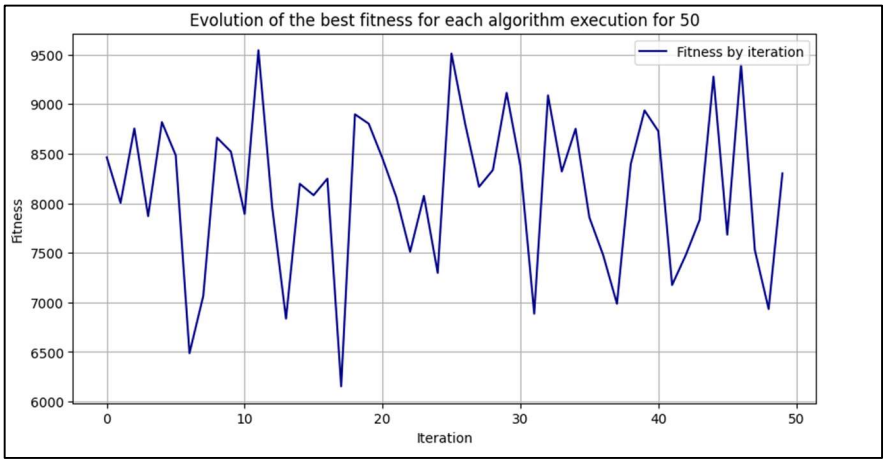
Results 3rd Case

```
Time of execution for a random execution of the algorithm:
0.3627972
Number of fitnesses calculated for an execution of the algorithm:
3000
```

Graph [1.1]: Evolution of the fitness every iteration for a random execution

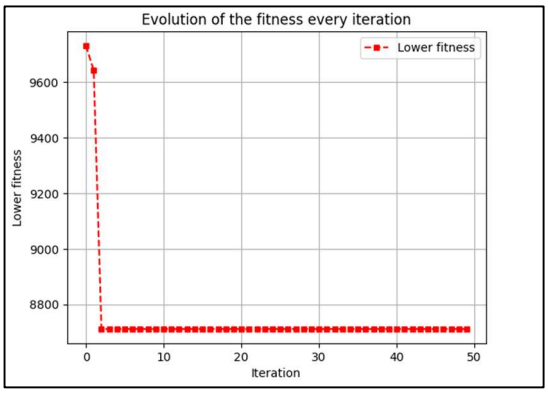


Graph [1.2]: Evolution of the best fitness for each algorithm execution for 50 times

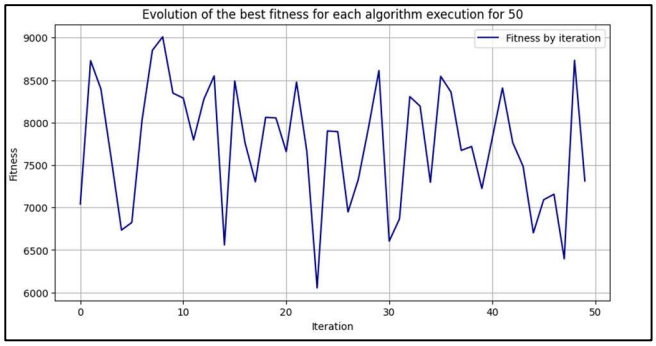


The low number of population size, generations and low probability of mutation makes a good opportunity to test the convergence of the different parameters of the tournament parent method (with replacement or changing the p parameter).

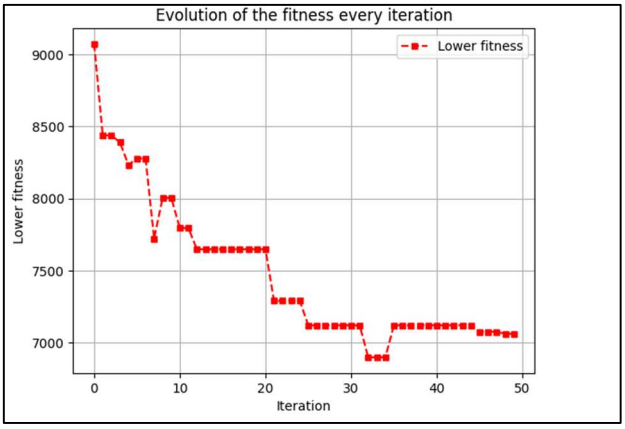
By changing the replacement parameter and setting it as False, we obtain a faster convergence as we evaluate less combinations.



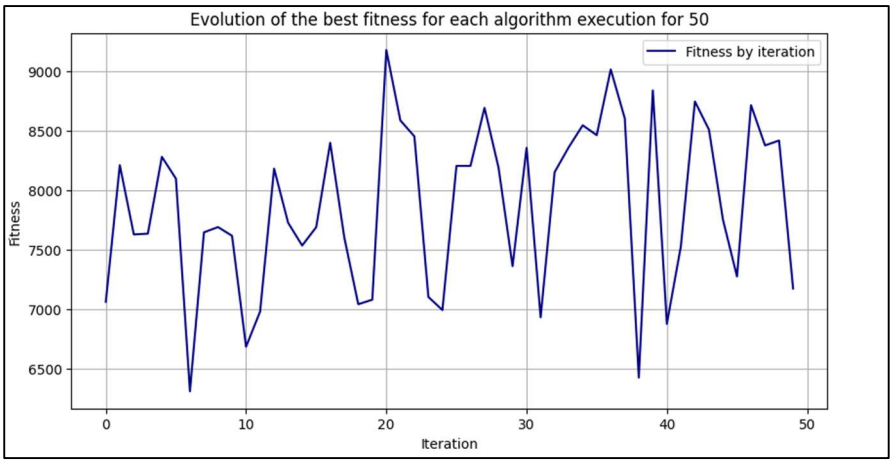
Moreover, the average results obtained are very similar to the ones with replacement, with a little improvement in general.



Also, changing the p to 0.5 and letting not always the best fitness of the tournament to win, affects significatively to the convergence



And the average results doesn’t experiment any change compared to the others



Interpretation of results:

As in the first random execution with, the best solution was:

```
['Toledo', 'Cáceres', 'Pamplona', 'Madrid', 'Coruña', 'Murcia',
'Valencia', 'Alicante', 'Córdoba', 'Cádiz', 'Huelva', 'Sevilla',
   'Barcelona', 'Girona', 'Tarragona', 'Zaragoza', 'Bilbao',
                'Donostia', 'León', 'Oviedo']
```

with fitness of 7527, we say the best route starts from Pamplona, continues to Madrid or Cáceres and then returns from the other side as is a circuit. As we can see

As we said before, the main goal of this case, which is very sensible to changes due to the low range values, was to evaluate the possibilities of the tournament method to see which configuration is the more beneficial for us. In this case we know that if we want to prioritize the convergence we should quit the replacement option and if we want to add more diversity maintaining the time of computation we should reduce the p probability.

At this point we are prepared to start doing some robust test, knowing how the algorithm works, to approximate a good result with the best configuration possible. We will consider the convergence, the diversity generation, the population size and the computation complexity evaluated above as the principal factors to find the best solution.

4th Case

Parent selection method: `Tournament Selection (k=2)`

Crossover method: `PMX Crossover`

Mutation method: `Shake Mutation (mutation_rate=0.1)`

Survivor method: `Genitor`

Iterations: `150 Generations`

Population Size: `80`

Times executed the algorithm: `50`

In this case, we chose the tournament selection method without replacement to favourise the convergence and the p = 1, decreasing the population diversity but priorizing the convergence. We also apply the PMX crossover to put respect the modifications in the order of the cities and we implemented a shake mutation method as before and we made a notable augmentation of the population size and the iterations

Results 4th Case

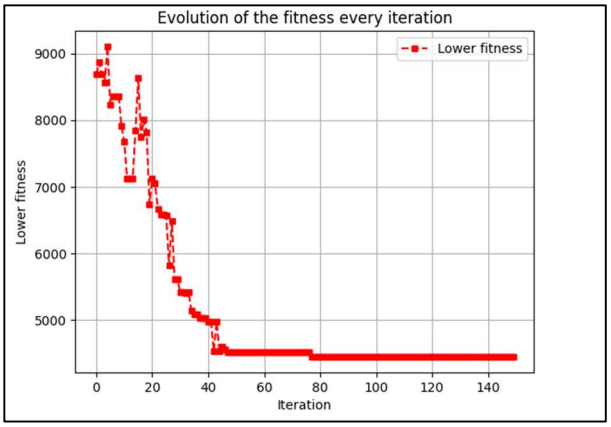
Time of execution for a random execution of the algorithm:

1.1620821

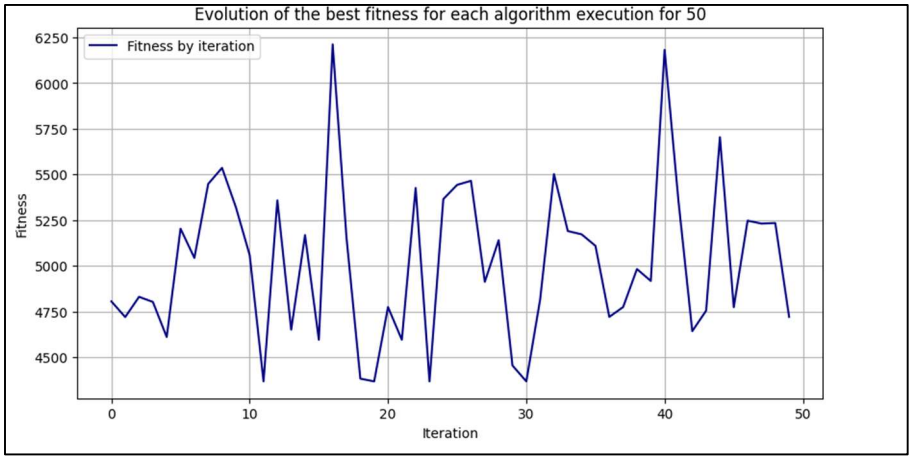
Number of fitnesses calculated for an execution of the algorithm:

24000

Graph [1.1]: Evolution of the fitness every iteration for a random execution



Graph [1.2]: Evolution of the best fitness for each algorithm execution for 50 times



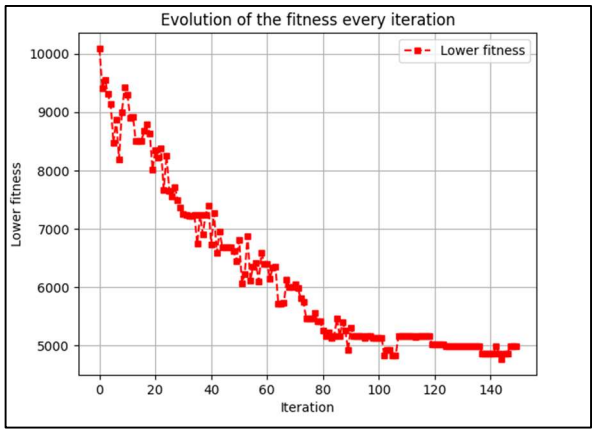
Interpretation of results:

As in one random execution, the best solution was:

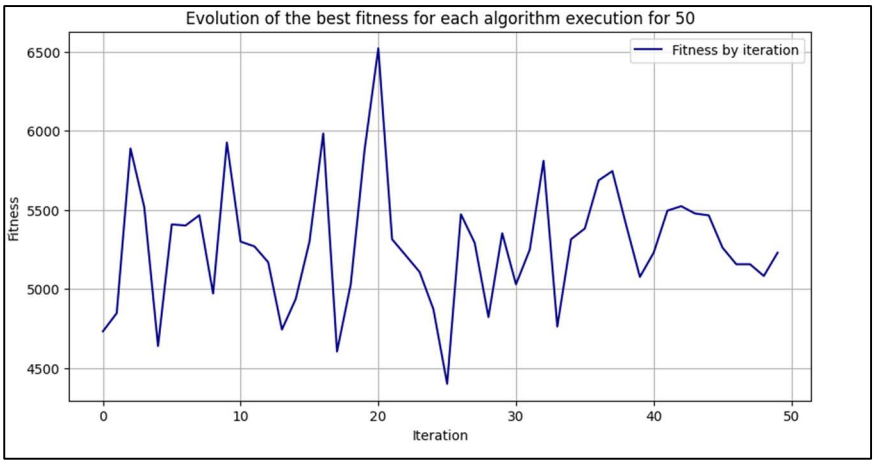
```
['León', 'Coruña', 'Oviedo', 'Bilbao', 'Donostia', 'Pamplona',
 'Zaragoza', 'Barcelona', 'Girona', 'Tarragona', 'Valencia',
'Alicante', 'Murcia', 'Madrid', 'Toledo', 'Cáceres', 'Córdoba',
                'Cádiz', 'Sevilla', 'Huelva']
```

with fitness of 4717, we say the best route starts from Pamplona, continues to Zaragoza or Donostia and then returns from the other side as is a circuit. We can see a combination between a fast convergence always achieved and good results in general, watching an average result of less than 5000 kilometers.

I also tried changing the parent selection method and put in value another time if a fast convergence is a positive aspect to take into account. When I tried with the roulette method I saw how the convergence is lower but also gets to good solutions.



Here is also the average result of executing the same configuration with the roulette method.



As we can see, the average result is not as good as in the tournament method so we will continue making little changes to this case, applying the tournament method till we find the best solution. Now, in the 5th case, we will decide if it's better the PMX or the Order-based crossover method.

5th Case

Parent selection method: `Tournament Selection (k=2)`

Crossover method: `Order-based Crossover`

Mutation method: `Shake Mutation (mutation_rate=0.1)`

Survivor method: `Genitor`

Iterations: `150 Generations`

Population Size: `80`

Times executed the algorithm: `50`

In this case, we combine a small k to add diversity to the population and quit from a fast converge and we implement the order-based crossover to maintain optimal sequences, and with shake mutation at a probability of 0.1 to introduce variability and avoid premature convergence.

Results 5th Case

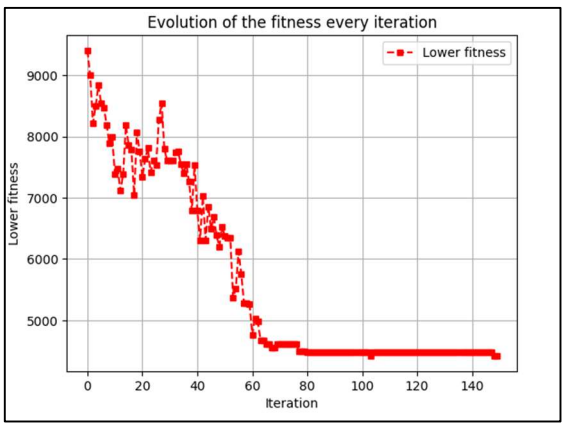
`Time of execution for a random execution of the algorithm:`

`0.900410`

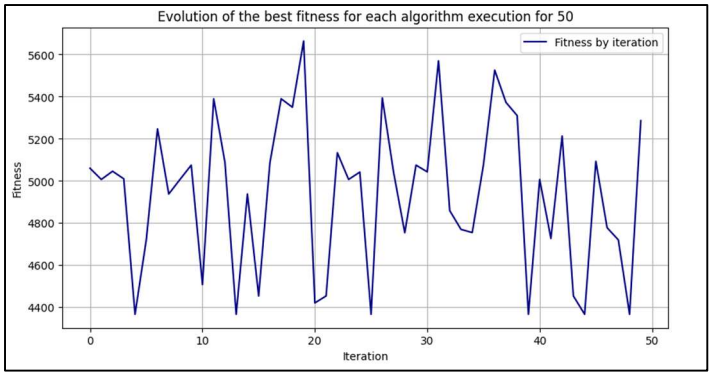
`Number of fitnesses calculated for an execution of the algorithm:`

`24000`

`Graph [1.1]: Evolution of the fitness every iteration for a random execution`



Graph [1.2]: Evolution of the best fitness for each algorithm execution for 50 times



Interpretation of results:

As in one random execution, the best solution was:

```
['Tarragona', 'Barcelona', 'Girona', 'Zaragoza', 'Pamplona',
'Donostia', 'Bilbao', 'Oviedo', 'León', 'Coruña', 'Madrid', 'Toledo',
'Cáceres', 'Huelva', 'Cádiz', 'Sevilla', 'Córdoba', 'Murcia',
'Alicante', 'Valencia']
```

with fitness of 4421, we say the best route starts from Pamplona, continues to Zaragoza or Donostia and then returns from the other side as is a circuit. We observe a slower convergence that could be compensated with a more diversity population evaluation, leading to a better result. In any case we are above 5700 kilometers and in many cases what seems to be the lowest fitness we can obtain, 4367. Moreover, the time of execution is lower and the solution appears to be near.

Some iterations, it seems the convergence is too fast and arrives to a good result but not the best, we will amplify the diversity by increasing the k and changing from shake mutation to another one which preserves an order but changing the positions, for example, the inversion method.

6th Case

Parent selection method: `Tournament Selection (k=5)`

Crossover method: `Order-based Crossover`

Mutation method: `Inversion Mutation (mutation_rate=0.1)`

Survivor method: `Genitor`

Iterations: `150 Generations`

Population Size: `80`

Times executed the algorithm: `50`

Results 6th Case

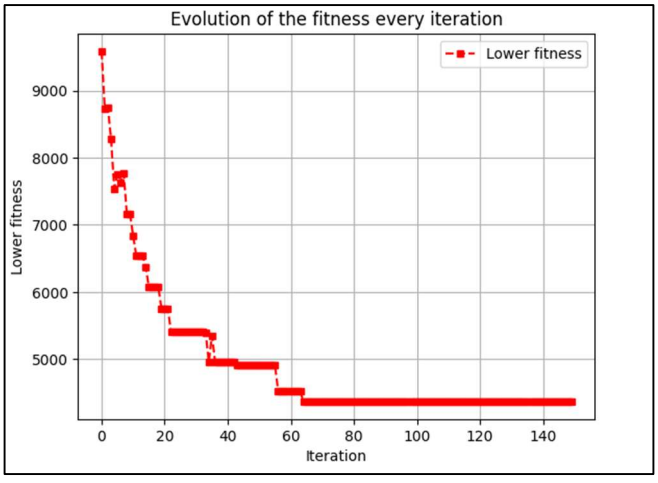
`Time of execution for a random execution of the algorithm:`

`0.9499068`

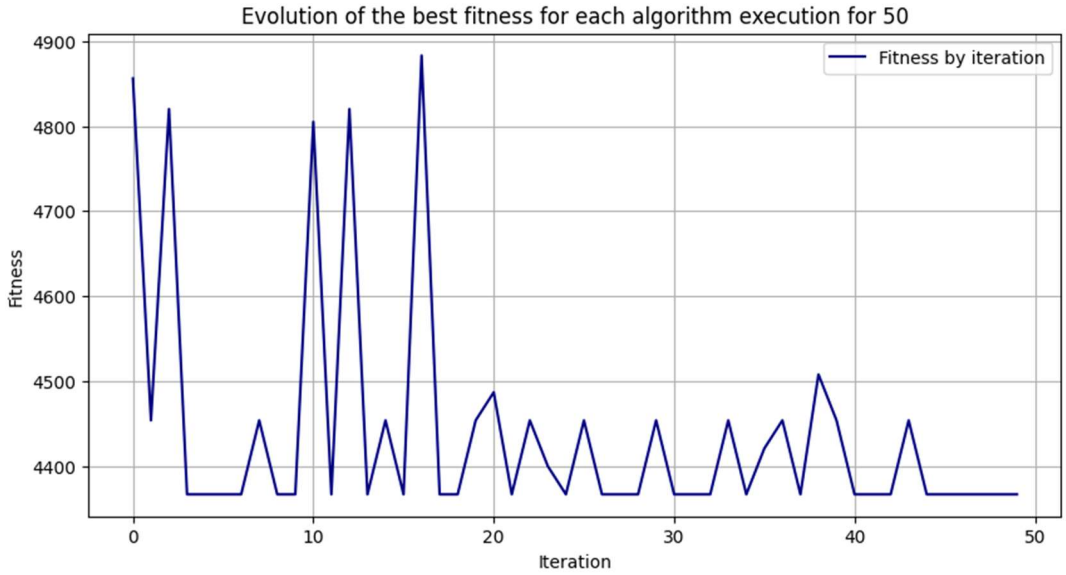
`Number of fitnesses calculated for an execution of the algorithm:`

`24000`

`Graph [1.1]: Evolution of the fitness every iteration for a random execution`



`Graph [1.2]: Evolution of the best fitness for each algorithm execution for 50 times`



Interpretation of results:

As in one random execution, the best solution was:

with fitness of 4367, we say the best route starts from Pamplona, continues to Zaragoza or Donostia and then returns from the other side as is a circuit. We observe a good convergence, evaluating a very diverse population and leading to a very good result. In any case we are above 4900 kilometers and in almost iterations, we found the best solution. Moreover, the time of execution is low and the fitness evaluation are as in the other cases.

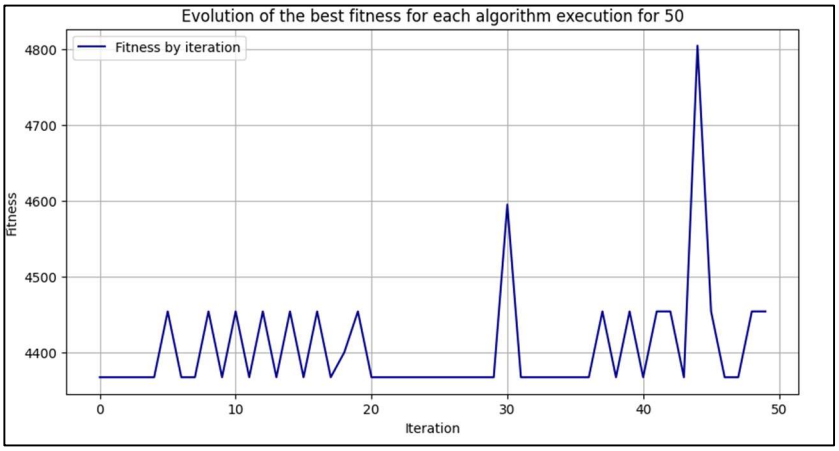
Conclusion

Having studied all the functions and after a few tests, I found that to find the solution, I only have to modify the size population, probability of mutation and the k participants in the tournaments. No the number of generations because in any case converges in a high iteration (more or less at the 80 iteration)

Depending on the aim we can obtain one solution or another.

If we want to obtain an accurate solution in the minimum time possible the last configuration reducing the number of generations to 80 (it's where almost all the solutions have already converged).

However, if we want to obtain the accurate solution always, a good idea is to increase the k participants (for example to 8) and the population size (for example to 150) and we will obtain an algorithm that, in the 96% of the times, obtains a solution under 4450 in an average time of 2 seconds.



As it was said before, the best solution obtained is that, to visit the given cities of Spain, we will have to travel at least **4367 km**. From Pamplona to Zaragoza, Barcelona, Girona, Tarragona, Valencia, Alicante, Murcia, Córdoba, Cádiz, Sevilla, Huelva, Cáceres, Toledo, Madrid, León, Coruña, Oviedo, Bilbao, San Sebastián and Bilbao to return another time to Pamplona and complete de circuit