# Evolutionary Algorithms: Group report

Suat Mert Altuğ, Georgios-Vissarion Foroglou, Nikolaos Papageorgiou

November 7, 2024

## 1 An basic evolutionary algorithm

### 1.1 Representation

After a thorough literature review and a couple of fruitful discussions between the members of the team and during the lectures, we decided that the most suitable representation for the TSP problem is one based in permutations. Permutations are ideal for representing TSP solutions because they inherently define an ordered sequence that visits each city exactly once, which matches the problems requirements. Another potential candidate that was discussed was binary encoding but it was rejected, due to the fact that it does not offer any advantage for the representation.

Regarding our representation, each candidate corresponds to an 1D numpy array of integers. The $i^{th}$ element of each of these arrays corresponds to the index of the city which the salesperson will visit in the $i^{th}$ step. The information about the city index is gained directly from the CSV files. The size of these arrays is $num\_cities + 1$ , where the first $num\_cities$ elements are the tour, and the last element is the copy of the first element to satisfy the property of a cycle. Finally, the population is a 2D numpy array where each row stores an individual.

### 1.2 Initialization

One of the biggest problems regarding the initialization of the population is finding an effective way of randomly obtaining feasible solutions. We can easily observe that the provided datasets contain graphs that are both fully connected and not. This might eventually lead to infeasible solutions since random permutations will most probably contain at least one pair of cities that are not connected.

Given that in the initial phase of the project the goal was to develop a functional evolutionary algorithm capable of finding a solution in the smallest dataset, we decided to follow a basic approach for initialization. To initialize the population we decided to just get random permutations - if the permutation is feasible then the candidate is accepted in the initial population. The algorithm was able to create a feasible dataset in an acceptable time period, therefore this process proved to be effective for the small dataset.

By testing this approach for the larger datasets, however, the curse of dimensionality was found to be a problem, as the probability of having pairs of not connected cities naturally gets larger as the length of the cycle increases. Designing a more sophisticated initialization algorithm will be important for the next phase. The most important challenge that we will need to face is creating a set of feasible solutions within an acceptable computation time without reducing randomness, exploration or diversity factors which are necessary for a good initialization in EAs. There are multiple more sophisticated approaches we could take to achieve this, like the use of search algorithms and heuristics (e.g. random walk or A*).

### 1.3 Selection operators

Selection and elimination algorithms were chosen simultaneously. We decided to not use the same technique for both steps, because it might become too biased towards good performing individuals (higher probability to pass through both of the selective steps), which might reduce diversity. During the lectures we discussed about two major categories of such algorithms, fitness based and competition based methods.

Elimination seems like a stricter process, as it can completely erase the information of one candidate for the next generations. On the other hand, selection should be less strict to give emphasis on the fact that not well performing candidates (in terms of obj value) should be allowed to give offspring, in order to increase diversity. As the generations go on, the strictness can increase, but in the early stages exploration is crucial.

Thus, for selection, we decided to go with a ranking based method. These types of methods have the advantage that it is fairly easy to control the strictness (through $s = \alpha^{gen\_num}$). We can start by choosing $a \simeq 0.99$ (chosen from the slides), since values near 1 are considered reasonable. We chose the quadratic method over the

linear, because the latter might have been too lenient on the later stages.

### 1.4 Mutation operator

A mutation operator was chosen based on chapter 4.5 of the recommended book. In that chapter, we found out that permutation problems can be classified in two categories, ”order” and ”adjacency” problems. Naturally, TSP is an ”adjacency” problem thus, according to the book, a suitable mutation method is an inversion operator. This operator follows a straight-forward approach; a random subtour of the candidate is chosen and it gets inverted. The outcome of this process is our mutated candidate.

We chose this operator because it is designed with a high degree of respect for the connectedness of the cities, therefore the probability of getting an infeasible mutated candidate is smaller than in the other techniques. In the implementation, if a candidate is chosen to be mutated, we start creating mutations until we find a feasible one (while this is a fairly naive approach it provided adequate results for this initial EA).

This algorithm generally does not induce a large amount of randomness (the output is not greatly dissimilar to the input). The algorithm also does not have any parameters. However, our preliminary results (paragraph 2.1) showcased a functional evolutionary algorithm even with this simple mutation operator.

### 1.5 Recombination operator

Like for the mutation operator, recombination was also based in the book. We found out that for problems like TSP, there are two classic choices: PMX and Edge crossover operators. Due to the report's page constraints we havent provided detailed descriptions of these algorithms, but the book contains much detail for both. On a first glance PMX seems simpler to understand and implement.

We implemented both of those techniques, and we observed that in Edge crossover the probability of creating an infeasible offspring is much higher than in PMX. Giving a detailed explanation to why this happens is a difficult task, but if we had to give an interpretation we would say that it might have to do with the fact that in our implementation the Edge crossover might have to pick an unvisited node randomly in cases when none of the neighbours have unvisited neighbours (and other similar details).

The aforementioned problem could probably be solved with a more sophisticated implementation which is however more suitable for the individual phase of the project. Given the above, as well as the fact that PMX provided more feasible offsprings, we decided to use the PMX crossover operator. PMX does not have adjustable parameters, and we did not use self-adaptivity. We demanded the parents to give offsprings with a probability=1, and if there are infeasibilities we just repeated the recombination operation.

### 1.6 Elimination operators

As we already mention in paragraph 1.3, picking a different type of algorithm in elimination compared to the selection step was crucial. Thus, for elimination, we used a competition-based algorithm, specifically k-tournament. We chose this one because it is simple to implement and it generally provides decent results (as we discovered on the hands-on programming session). A smaller k (e.g. 3) applies a smaller selection pressure than a larger one (e.g. 10). The reasonable values for k are generally in between and near these values. We should highlight the fact that we have included elitism (i.e. the most fit individual always survives) to make sure that we do not lose the best solutions.

### 1.7 Stopping criterion

There are multiple interesting stopping criteria that could be used for our EA. For example, we could demand that if the mean objective value is similar to the best objective value then the algorithm should stop (because we have lack of diversity and the algorithm has converged). Another approach could be to work with the differences or relative differences of the objective values between consecutive generations (not demand too small of a difference though, these algorithms do not excel in converging to (local) minima with high precision). However, for this initial phase of the project, we decided to use a simpler but adequate approach; a fixed number of generations for the algorithm to run.

## 2 Numerical experiments

### 2.1 Chosen parameter values

Table 1 shows information about the choice of parameters. We chose Pop_size=1000, Num_offspring=Pop_size, k=3, Mut_prob=0.05, $\alpha$=0.99, total_generations=150.

| Name | Range | Why this choice | Effects of different values |
|---|---|---|---|
| Pop_size | (100, ... ,1000) | Similar Values found in Book Examples | Larger value increases computation, improves solution quality, makes convergence gradual. |
| Num_offspring | (0.5, ... ,1)*Pop_size | Examples, Trial and error | Increasing causes increase in computation time and faster convergence. |
| k | (3, ... ,10) | Lectures, Examples | Larger value increases selection pressure |
| Mut_prob | <0.05 | From Lectures | Increases diversity but slows convergence |
| $\alpha$ ($s = \alpha^{gen\_num}$) | (0.95, 0.99, 0.999) | 0.99 from lectures, others from experiments | Lower means larger selective pressure, especially in the earlier stages |
| total_generations | (100, ... ,500) | Examples, Trial and Error | Increases computation time but smaller population might not suffice for convergence. |

Table 1: Decision Process on Parameters

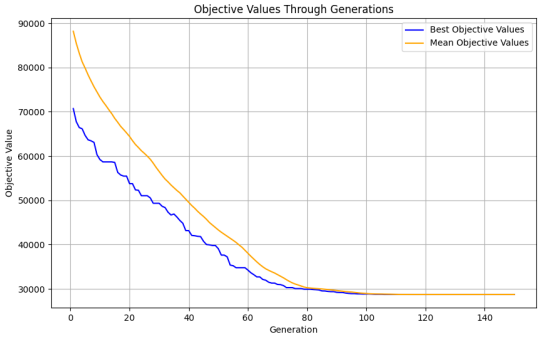


Figure 1: Tour 50 Results, t=175s, Optimal=28729,57

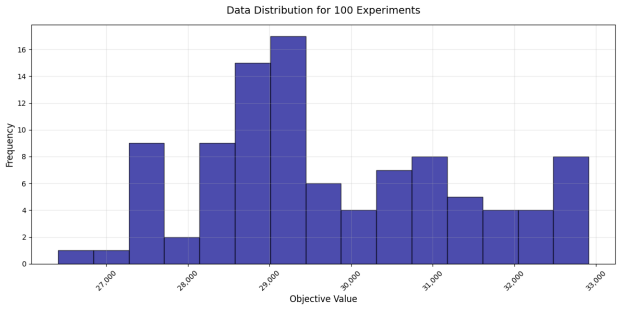


Figure 2: Histogram of Best Values (100 experiments)

### 2.2 Preliminary results

The results for the experiment on the small dataset can be seen on Figure 1. The program run for 150 generations and it converged around generation 120, so needed around 140s for convergence. In terms of time complexity the algorithm could be better, as we do not use in-place operations (creating new arrays in every iteration) or any sophisticated method to reduce the objective evaluations and most importantly we are not using any special method to handle infeasible solutions yet. The goal of this phase was simply to have a running algorithm able to solve the small problem, these improvements will take place on the competitive individual phase.

In terms of memory we do not use any algorithm that increases the space complexity, so basically it is defined by the 2D arrays of the populations and/or the adjacency matrix (depending on which has more rows).

The algorithm converges relatively fast (before 150 generations). This could be attributed to the fact that the problem is fairly small. For the diversity, we can definitely say that after generation 100 the mean objective value stays close to the best value, which shows that the diversity has been lost. In the beginning, the population should be more diverse, due to random initialization.

To benchmark our algorithm works correctly, we decided to run it on the dataset with the 100 cities. We get the subdataset of the first 50 cities and apply the algorithm with the same parameters. Here the graph is fully connected and metric. When a graph has these two properties, then we can apply the Christofides algorithm, and we know that we will have a 3/2-approximation[1]. The Christofides algorithm gives an answer of value **65053.23**, while our algorithm gives **59850.83**. This shows that our result is not larger than 1.5 times the actual optimum and therefore is a sign that our algorithm is acceptable (at least on fully connected problems).

The best value we found in all the experiments was **26409,4447**. It is difficult to determine with certainty if this is the global optimum for the problem, but an educated guess would be that its not. Even though we believe that our algorithm works decently, evolutionary algorithms are not guaranteed to find the global optimum. We would guess that we should make further improvements (one might be to apply extensive local search on the result to find the nearby local optimum) to increase the chances of finding the global optimum, this initial approach of the group phase is maybe too simplistic.

By repeating the experiment 100 times, we get varying results which totally makes sense because the algorithm is stochastic in nature. Some descriptive statistics for the final objective values found are: $mean = 29747.406, std = 1575.054, min = 26409.444, max = 32909.254$.

## References

[1] Christofides, N. Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem. *Operations Research Forum* **3**, 20 (2022). URL `https://link.springer.com/10.1007/s43069-021-00101-z`.