Evolutionary Algorithms: Group report

by Davide Castellucci, Ioannis Mouratidis and Cedric Mousin

1. Introduction

Genetic Algorithms and Evolutionary Computing grade is based on a project implementing an evolutionary algorithm for the traveling salesperson problem in Python 3.8.6.

The traveling salesperson problem consists of minimizing the length of a cycle that visits all vertices in a weighted, directed graph.

In this project, three distinct datasets are used to evaluate the evolutionary algorithm:

- *tour29.csv*
- *tour194.csv*
- *tour929.csv*

They are defined by their content from the slimmest dataset (*tour29.csv*) to the largest (*tour929.csv*) with an intermediate dataset (*tour194.csv*).

2. An elementary evolutionary algorithm

2.1. Representation

The representation reflects the definition of individuals contained in the population. This process implies to represent the data from the problem into the evolutionary algorithm in such a way that it can process them [1]. The representation of each individual candidate solution is done with a numpy array representing a permutation of the cities in the order they are visited. The total set of solutions is represented by an array of arrays. Numpy array has been chosen as this data type mainly due to its ease of use. Another option, if the verification of inclusion of a specific element in the population is required multiple times for the algorithm, is the use of a data object with built-in hashing, such as a set, due to the faster implementation of the "in" operator.

2.2. Initialization

The initialization step is the first block contained in the evolutionary algorithm framework. The first approach is to generate randomly individual candidate solutions based on the

original population. Even in the initialization phase, there is a trade-off to find between taking the entire population or defining an initial population with a higher potential related to our problem. Introducing additional effort to better shape the initialized population leads to potential computing time [1]. In the salesman problem, the random feature is therefore a permutation of the cities into the array. The variance of the randomness has been checked to validate that there is no redundancy in the initialization phase. Throughout multiple tests, the variance of the objective value of the population was calculated to illustrate that, for a first approach, the diversity obtained is sufficient.

A second step is to consider other initialization mechanisms to optimise the time required by the algorithm to converge. For test samples named tour29.csv the algorithm converges in a reasonable time. For the other data sets, further optimisation will be required in the next phase.

2.3. Selection operators

The selection operator is a significant step because it selects specific individuals from the initial population defined by the initialization phase. The selection step increases by its own nature the mean quality of solution in the population [1]. The mechanism chosen for the salesman problem is the k-tournament selection as a first approach. The k-tournament selection implemented samples randomly the initialized population with a defined number of individuals (k) and this procedure is repeated (l) times [2].

The option of k-tournament selection is often a solid starting point [2]. The objective is also to get enough flexibility to test several parametric impacts like the number of candidate solutions (k) and the iteration number (l). If the number of candidate solutions is too restrictive, there is also a possibility that a good potential parent is not taken into account. The k-tournament selection is also more flexible concerning the use of distinct recombination operators and even some of them can combine more than only two elements.

The roulette wheel selection has also been considered during the discussion about the salesman problem. Due to the strength and the easiest implementation of the k-tournament selection, the roulette wheel option has not been investigated further.

2.4. Mutation operators

The mutation or variation is the key operation in the evolutionary algorithm framework as it defines the trade-off between the exploration and exploitation. The mutation operator must be carefully selected to emphasize the most promising features of the selected candidates [2]. As a first step, the swap mutation has been chosen to randomly swap the order of two cities in the array with a probability of 5%. The intent is to introduce enough variance in the original array to improve the solution if a parent is already close to the optimal. After

some consideration, the mutation has been applied to both parents and children since there was no particular reason to apply it to children only.

The swap mutation coefficient probability of 5% was working adequately but the algorithm often returned a local optimum. Doubling the probability coefficient helped to avoid convergence to the local optimum. The swap mutation coefficient is fixed for the entire algorithm, but an improving way considered is to implement a variation of the mutation. Increased values of mutation coefficient at the beginning of the process to widen the searching area and gradually lowering it to converge towards the solution. This feature requires to implement a parameter inside the function evolving automatically as the process converges towards the solution.

Several other mutation operator types have been considered, such as an inversing mutation. One such operator of particular interest, would be a variable mutation permuting for each element, a number of cities related to its fitness coefficient. In such scenario, the candidate solutions with a lower fitness coefficient would get a higher number of mutation steps to potentially increase their viability as candidate solutions. Another idea was to implement different types of mutations, not only changing the mutation probability, but also the mutation type as the algorithm approaches the result. This could allow starting with a mutation that performs more drastic changes and then switch to a more minor mutation that only explores the local search space, such as the swap mutation. This is a potential area of optimisation that must be considered in the future.

2.5. Recombination operators

The recombination step has the objective to merge specific information from each parent into the children generated [1]. The recombination operator is defined to maintain the most interesting features of individuals but still with randomness.

The candidate solutions from the mutation steps are represented as permutations inducing that the recombination operators for permutations must be taken into account. First two potential permutation operators have been considered: order crossover and cycle crossover. To determine which operator is the most appropriate, some search into the literature and based on the findings of P. Larrañaga et al [3], the implementation of order crossover over cycle crossover has been done.

At the early stage of the algorithm development, the permutation operator was only applied once per set of parents, generating one child. To improve the genetic pool and the diversity, the second step was to generate two children from the same set of parents. Adding a second child allowed the algorithm to converge faster. Generating even more children per pair of parents, using various methods, is an interesting way of investigation that can be done in later development stages.

Based on the literature [3], implementing different recombination operator, such as the Edge Recombination Operator could improve the results obtained. This, as well as other recombination operators could be an interesting way to test the efficiency of several recombination operators applied to different matrix sizes. This is also one of the potential future development and optimization phases considered for the algorithm.

2.6. Elimination operators

The evaluation or elimination step acts as the finest filter in the entire evolutionary algorithm. The selected population (called parents) added to the created population (called children) can lead to a significant increase of potential candidate solutions. To prevent this issue, a fitness-based elimination is implemented based on the (λ+μ)-elimination. The (λ+μ)-elimination process merges both parents (λ) and children (μ) population into a single set of candidate solutions (λ+μ). From this new set of candidate's solutions only the top ranked candidate solutions are considered for the next step.

During the testing phase of our evolutionary algorithm, the elimination operator based on (λ+μ)-elimination was not the most appropriate regarding our problem. The candidate solutions often bring duplicate solutions not being eliminated, narrowing the searching area of the algorithm being stuck in local minima. An option investigated is to delete the duplicated solutions, but it has not been efficient and increases the computation time of the algorithm to converge.

Another optimization step is to explore additional elimination strategies or a combination thereof to check if this issue can be solved or at least to get a better trade-off between the computation time and the solution candidates evaluated. One such example considered for future implementation was using a combination of elitism and a fitness proportional selection like keeping some of the best individuals for the next population and adding some randomness for the rest of the "spots", while still favouring the best.

2.7. Stopping criterion

The implementation of a stopping criterion is mandatory. There is no guarantee to find the optimum solution because the evolutionary algorithm is stochastic. It induces that as the optimum cannot be reached, without stopping criterion, the algorithm enters an infinite loop [1]. The definition of a stopping criterion can be done on several parameters like:

- Limitation of the iteration number if the fitness of the candidate solutions does not improve after a few iteration
- Time limitation if the allowed time to extract a candidate solution is limited
- Limits of the population diversity drops inducing that the same candidate solutions are represented

The ability of the algorithm to find a solution close to the optimal must be tested. With the smaller dataset (*tour24.csv*), repeating a fixed number of iteration (initially 10, later 50) without any improvement to the best solution found gives a good indication that the local minimum has been reached. To allow this solution to be more scalable, a diversity function has been implemented calculating the variance of the population. This will then be considered as part of a stopping criterion for larger datasets.

Knowing that the final phase of the project is to evaluate the time required by the algorithm to find a solution, another time-based stopping criterion can be implemented through the Reporter module (*Reporter(1).py*). This additional feature can be implemented on a later development stage to improve the algorithm and evaluate its impact. The objective is to provide an algorithm combining variance and time as stopping criteria.

Depending on the size of the dataset, considering other stopping criteria can be done depending on the time available to implement it.

3. Numerical experiments

3.1. Chosen parameter values

The chosen parameter values, derived from the numerical experiments are an initial population of 100 individuals, a generation of 200 offspring every cycle, a selection of 5 candidates for k-tournament selection, a probability of mutation of 10% and, as a stopping criterion, the necessity to obtain the same result 50 times in a row. Further details related to the definition of those thresholds are illustrated in the next sections.

3.2. Preliminary results

We tried to find empirically the best hyperparameters by running the algorithm several times changing one parameter at a time and taking the average of the result in terms of quality and speed in order to see how the results varied.

The initial parameters were a population of 50 individuals, a generation of 100 children, a probability of mutation of 10%, as a stopping criterion finding the same objective function 50 times in a row and 5 random candidates parents selected in k-tournament selection.

	Initial parameters
Population	50
Candidates parents selected by k-tournament	5

Children generated	100
Mutation probability	10%
Stopping criterion	50 times in a row

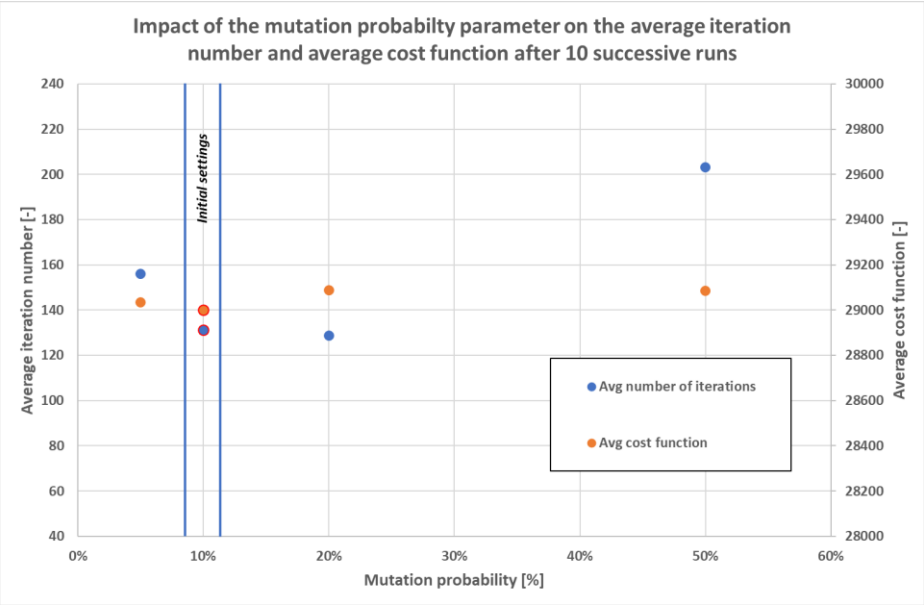
Based on those initial setting used as a reference, ten successive run of the algorithm have been launched and the average number of iterations as well as the average cost function have been monitored in order to evaluate their impact. The first parametric check focused on the number of children generated but also the number of parents selected. The results from the tests performed are illustrated in the following table:

		Initial parameters	*Parametric analysis*			
			Case a	*Case b*	*Case c*	*Case d*
	Population	***50***	***50***	***100***	***100***	***200***
	Candidate parents selected by k-tournament	***5***	***5***	***5***	***5***	***5***
	Children generated	***100***	***200***	***100***	***200***	***400***
	Mutation probability	***10%***	***10%***	***10%***	***10%***	***10%***
	Stopping criterion	***50 times in a row***	***50 times in a row***	***50 times in a row***	***50 times in a row***	***50 times in a row***
Average value after 10 runs	***Avg number of iterations***	***164,6***	***134,2***	***197,7***	***131,1***	***124,9***
	Avg cost function	***31391,8***	***29823,7***	***29968,6***	***28999,8***	***28450,5***

Increasing the number of individuals and generating offspring highlights some improvements but this is not as big as the amount of time that the algorithm has to spend to perform every cycle. This cannot be noticed very well with the dataset *tour29.csv* but with *tour194.csv* and *tour929.csv* it became really evident, so the best trade off choice regarding time and results seemed to have an initial population of 100 individuals and a generation of 200 children every cycle.

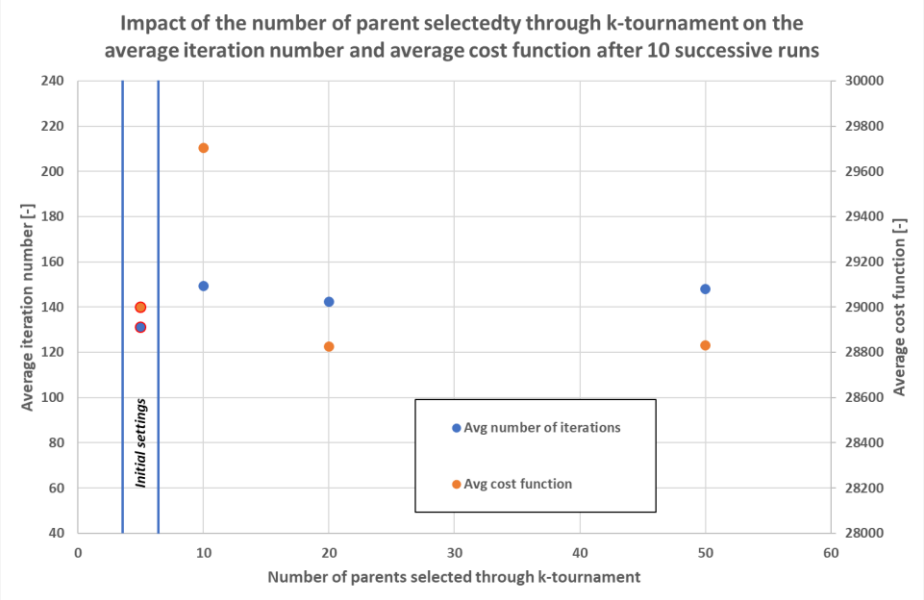
Maintaining these parameters depicted in Case C, the focus has been done on the other parameters relying on the simplistic assumption that they are independent to the ones found so far. Case C being now the initial parameters, the first study focused on the probability mutation and its impact. The original values of 10% has been changed to 5, 20 and 50% and the results are depicted in the table and plot below:

		Case c	Parametric analysis		
			Case c-1	Case c-2	Case c-3
	Population	100	100	100	100
	Candidate parents selected by k-tournament	5	5	5	5
	Children generated	200	200	200	200
	Mutation probability	10%	5%	20%	50%
	Stopping criterion	50 times in a row	50 times in a row	50 times in a row	50 times in a row
Average value after 10 runs	Avg number of iterations	131,1	156,1	128,6	203,1
	Avg cost function	28999,8	29035,2	29087,8	29086,4



Still using Case C as the initial parameters, the second focused on the number of candidate parents selected during the k-tournament and its impact. The original values of 5 has been changed to 10, 20 and 50 and the results are depicted in the table and plot below:

		Case c	Parametric analysis		
			Case c-4	Case c-5	Case c-6
	Population	100	100	100	100
	Candidate parents selected by k-tournament	5	10	20	50
	Children generated	200	200	200	200
	Mutation probability	10%	10%	10%	10%
	Stopping criterion	50 times in a row	50 times in a row	50 times in a row	50 times in a row
Average value after 10 runs	Avg number of iterations	131,1	149,4	142,3	148,1
	Avg cost function	28999,8	29703,4	28825,6	28830,9



With a full deterministic selection of the individual (namely chose the best 2 every time – Case c-7) we obtain an average number of iterations equal to 130.5 with an average cost function of 29532.5

Finally with a fully random selection (Case c-8) we obtain an average number of iterations equal to 135.3 with an average cost function of 29746.9.

		Case c	Parametric analysis	
			Case c-7	Case c-8
	Population	100	100	100
	Candidate parents selected by k-tournament	5	100	2
	Children generated	200	200	200
	Mutation probability	10%	10%	10%
	Stopping criterion	50 times in a row	50 times in a row	50 times in a row
Average value after 10 runs	Avg number of iterations	131,1	130,5	135,3
	Avg cost function	28999,8	29532,5	29746,9

Looking at these results we decided to keep the initial value of 5 candidates selected to have a good compromise between randomness time and results.

We tried to change the termination condition namely the number of times that we must obtain the same cost function compared to the initial parameters illustrated in Case c.

		Case c	Parametric analysis	
			Case c-9	Case c-10
	Population	100	100	100
	Candidate parents selected by k-tournament	5	100	2
	Children generated	200	200	200
	Mutation probability	10%	10%	10%
	Stopping criterion	50 times in a row	10 times in a row	100 times in a row
Average value after 10 runs	Avg number of iterations	131,1	82,3	179,7
	Avg cost function	28999,8	29466,4	28988,2

Also in this case it can be noticed how the solution tend to improve incrementing the number of consecutive results needed but the best trade off choice (especially for bigger dataset) seems to be the initial value of 50

3.3. List of identified issues

Through the development of the algorithm as well during the discussion several issues came out:

- Identified issues are the robustness of the code depending on the population size. It has been observed that the code was working pretty well for the smallest population in the appropriate amount of time. But this code was not performing well on the largest population (*tour194.csv* and *tour929.csv*).
- Variance of the randomness must be checked to avoid getting stuck in the local optimum. Being stuck in the local optimum can be solve either by changing initial population, elimination features or mutation characteristics.
- It also appears that same set of parents occurs after several iteration.
- The algorithm has not been tested yet on the largest dataset (*tour929.csv*). Therefore may be the initial settings found might be adapted again to fit all the three datasets.
- Datasets with disconnected cities are not properly handled, as they were no examples initially. After internal discussion at the start of the project, we concluded that, we most likely would assign the value "infinity" to invalid paths and allow the algorithm to eliminate those from the population. It would be interesting to observe the results of this method for very sparse datasets and if an adjustment would be needed.

4. Conclusion

The genetic algorithm related to the salesman problem allows to investigate each step of the genetic algorithm framework in details. The theory fixed the fundamental elements but there is still a need to do empiric check to define appropriate settings. Several ways of improvement have been defined even if the objective was to propose an algorithm dealing with several trade-off like the exploitation and exploration, the computing time and the number of iterations performed but also on several dataset.

5. References

[1] Introduction to Evolutionary Computing – Second Edition - A.E. Eiben and J.E. Smith

[2] Evolutionary Algorithms – Lecture notes – Prof. dr. N. Vannieuwenhoven

[3] Larrañaga, P., et al. “Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators.” Artificial Intelligence Review, vol. 13, no. 2, 1999, pp. 129-170.ResearchGate, https://www.researchgate.net/publication/226665831_Genetic_Algorithms_for_the_Travelling_Salesman_Problem_A_Review_of_Representations_and_Operators.