Evolutionary Algorithms: Final report

Paulina Rocha (r0869271)

May 8, 2024

1 Metadata

- **Group members during group phase:** Marius Zoller,Theresa Seidl, Venkatesh Viswanathan Manikantan, Paulina Rocha
- **Time spent on group phase:** 10 hours
- **Time spent on final code:** 60 hours
- **Time spent on final report:** 10 hours

2 Peer review reports (target: 1 page)

2.1 The weak points

1. Scramble mutation is too intrusive and not ideal. The amount of randomness generated by scramble mutation can significantly damage the current candidates.
2. Size of population and offspring scales badly with larger number of cities. The code from the group phase implemented a population size of 5 times the number of cities and an offspring size 5 times the population size, making it too time consuming for a larger number of cities.
3. K-tournament not selective enough. The k parameter set to two is not selective enough and cannot guide the population.
4. Too strict convergence rules. The limit of 300 iterations, 5 minutes, and an improvement lower than 1 % as stopping criteria are too strict. If there's still time and room for improvement, it seems bizarre to stop the algorithm after 300 iterations.

2.2 The solutions

1. Swap mutation was implemented in order to introduce diversity but not hinder too much the solution search as it was done before with the scramble mutation
2. Population size was set in accordance to the number of cities where 600 was used in problems with 500 cities or less, and 500 when above 500. The offspring size was set to two times the population size given that a larger offspring size could cause early convergence given the use of the local search operator.
3. The number of iterations as a stopping criterion was removed as the time and improvement criteria was enough to stop the algorithm.

Weakpoints not addressed:

1. The k parameter remained low as the local search operator 2-opt was implemented and increasing the k parameter value would have increased the selection pressure too much and diversity would have been lost.

2.3 The best suggestion

The team of Deniz Alp Savaskan, Deniz Can Oruc, and David Debot chose python lists to store their path representation when there are other alternatives which are tens of times faster when performing operations and use much less memory as well. For example the use of the package numpy can help with memory and computational time which will be of great help when solving problems with greater number of cities.

3 Changes since the group phase (target: 0.5 pages)

1. In the previous algorithm, a generational approach was implemented where only offspring selected through elimination would go on to the next iteration. The new algorithm now combines the original population

and the offspring, letting the survivors of the elimination go on to the next iteration.The change was made in order to increase diversity by giving another chance to the candidates that were not chosen as parents.

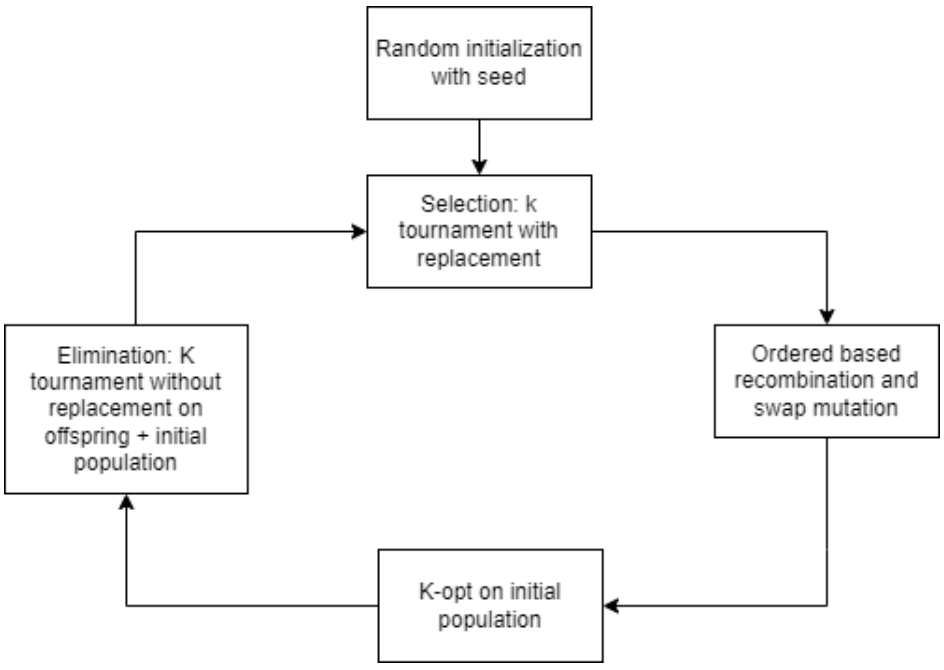
2. Implemented a local search operator in order to guide the exploratory process, which is of great help when dealing with a large search space due to a greater number of cities. The search operator was implemented randomly on 3 candidates of the population in order to help them since they did not benefit from the sharing of good genes that happens through the recombination operator. Given that the local search operator *2-opt* has great computational complexity, only 3 candidates chosen randomly are benefited.
3. The distances that were infinite between cities were replaced with a large number given that comparisons of fitness values cannot distinguish paths with 1 or more infinities.
4. Swap mutation was implemented in order to introduce diversity but not hindered too much the solution search as it was done before with the scramble mutation
5. Population size was set in accordance to the number of cities where 600 was used in problems with 500 cities or less, and 500 when above 500.
6. The offspring size was set to two times the population size.
7. The k-tournament elimination was changed to without replacement since a tournament with replacement is detrimental to the evolutionary algorithm decreasing the time to convergence and diversity.

4 Final design of the evolutionary algorithm (target: 3.5 pages)

4.1 The three main features

1. The local search operator 2-opt
2. K-tournament selection and elimination.
3. ($\lambda + \mu$) go to the elimination phase

4.2 The main loop



4.3 Representation

A path representation was chosen that simply enumerates the cities and the indices are the order they are visited. The connection from the last city to the first city is implicit. Each solution candidate is then a 1D numpy array with length of the number of cities in the problem. The whole population is represented by 2D numpy array with the first axis representing the population size and the second the number of cities.
This representation was chosen since it is very intuitive to understand and work with. This also makes it easier to design appropriate variation operators and there is a lot of research based on this representation, so rich literature exists.

Initially a representation of the solution path as the adjacency matrix of the resulting graph we also considered. This would lead to a sparse matrix, where each row and each column must contain exactly one 1 and zeroes otherwise. This is because each city is visited exactly once.
This representation has the advantage that vectorized operations are easier to implement which saves time. The main disadvantages are the large amount of needed memory, and the difficulty in choosing appropriate crossover operators.

4.4 Initialization

The candidates are initialised with a random permutation of the city numbers by generating an enumeration of the cities and then shuffling it. This guarantees to generate only valid candidates. Since solutions to the TSP are shift invariant due to representing a full cycle, the same solution can be represented by multiple candidates. For a problem with 29 cities, there are 29 different ways to represent the same solution, since each city can be chosen as a start. To eliminate this multiplicity, the last city was fixed as starting city and just create random permutations of the rest. After initialization, a random candidate is chosen and the local search operator is applied in order to **seed** the initialized population. The population enrichment scheme does not immediately take over the population given that a low k parameter value is set for the k-tournament selection and there is a good population size. The local search operator is applied to three random candidates of the initial population, due to computational cost, before proceeding to the k-tournament elimination phase in order to give them a better chance in competing.

4.5 Selection operators

K-tournament selection was chosen due to the computational efficiency of the operator. The fitness value for each candidate only need to be calculated once and can then generate large offspring generations. Roulette wheel based solutions rely on a knowledge of the entire population. Thus, whether they are based on the fitness values or ranking, these selection mechanisms require value-scaling or sorting, which is computationally expensive for large population sizes. K-tournament natively provides a good diversity of the offspring population, since there are many tournament groups that do not contain the best individuals. In that way, weaker individuals have the opportunity to pass on genes.

Low k parameter value of 2 was chosen in order to have a low selective pressure and further promote diversity, balancing the effect of early convergence caused by the local search operator. The k-tournament selection was implemented with replacement to reduce selective pressure given that if a tournament without replacement is introduced, the least fit members of the tournament can never be selected.

4.6 Mutation operators

Swap mutation was implemented which works by swapping the order of two elements in the candidate's path randomly and only takes the mutation rate as a variable to perform the mutation. Scramble and inversion mutation were also tried but they were too intrusive and just hindered the evolutionary algorithm. On the other hand swap mutation was able to introduce diversity without hindering the search. The mutation rate is adaptive where it is set to 0.20 unless there has not been any improvement for 15 iterations, in that case it is set to 0.30.

4.7 Recombination operators

The TS-problem underlies certain base-conditions: all the elements of m have to be represented (thus all of the cities have to be present in the solution), and none of the elements shall be represented twice (thus none of the cities shall be visited twice, e.g. have two occurrences in the path). Given these preconditions, operators that maintain the relative or absolute order in the path are required.

Order based recombination (OX) operators are preferable since we mostly care about the adjacency and relative order of the cities and not the absolute ones. With this regard, OX is a good choice given it performs well in terms of processing speed and is "relatively" easy to implement. In order based recombination, the offspring

inherits the values between two crossover points from a parent in the same order and position. The rest of the elements are inherited from the other parent in the order they appear, skipping the elements already in the offspring. Thus the offspring inherits the order,adjacency and absolute position from one parent and inherits the relative order of the remaining elements from the other parent. If there is little overlap between the parents, some of the information of the relative order will be preserved from the other parent. In the ordered crossover implemented, the crossover points are randomly selected and its applied to all the parents.

4.8 Elimination operators

The population and offspring are joined and K-tournament selection is implemented. This way the population has a second chance to go over the new iteration, and in order to preserve diversity, a low k value of two was used. K-tournament was again chosen for its computational efficiency even though $(\lambda + \mu)$ elimination was also implemented but was discarded due to lack of diversity and early convergence.The k-tournament was implemented without replacement given that implementing a tournament with replacement is detrimental to the evolutionary algorithm causing early convergence and loss of diversity.

4.9 Local search operators

The local search operator 2-opt was implemented in the population that did not go through the recombination operator and could not benefit from the exchange of good genes. This way they are given a push to become better candidates and pass to the next iteration. Given the computational cost, only 4 candidates were chosen randomly to have the local search operator applied to them. The idea behind the 2-opt operator is two reorder pair of edges and implement the new path if the fitness values is shorter than the original one. A simple swap mutation was also implemented where adjacent nodes are swapped and updated if the new path has a better fitness value. However, the latter was discarded given that it did not make much improvement and increased the computational cost.

4.10 Diversity promotion mechanisms

Given the high computational complexity of the local search operator, a variation of the *Crowding* algorithm was tried as it is a simple algorithm. In this version of the crowding algorithm, instead of removing the candidate, the mutation operator swap is applied. This diversity promotion was discarded due to too much hindrance of the solution search.

4.11 Stopping criterion

The stopping criterion for the evolutionary algorithm is time of 5 min. A no improvement criterion for the best objective value was implemented where the evolutionary algorithm would stop after 60 iterations of no improvement of at least 0.005.

4.12 Parameter selection

The population size was set to 600 for problems with up to 500 cities and 500 for problems up to 1000 cities to start with good diversity but taking into account the computational cost.

The k-tournament parameter was set to two to maintain diversity given the use of the local search operator causing early convergence. Due to early convergence of the local search operator, the offspring size was set to two times the population size given that the offspring size is used to reduce the number of iterations to reach an optimal solution without too much computational cost.

The rate of mutation was adaptive where it was set to 0.20 percent but increased to 0.30 if there was no improvement of the best objective value of at least 0.005 after 15 iterations.

4.13 Other considerations

Parallel implementation was tried but could not be implemented due to time constraints.

5 Numerical experiments (target: 1.5 pages)

5.1 Metadata

The population size is based on the number of cities where it is 600 for up to 500 cities and 500 for up to 1000 cities. The offspring size is two times the population size. The mutation rate is adaptive where it is set to 0.20 unless there has been no improvement of at least 0.005 for 15 iterations where it is set to 0.30.

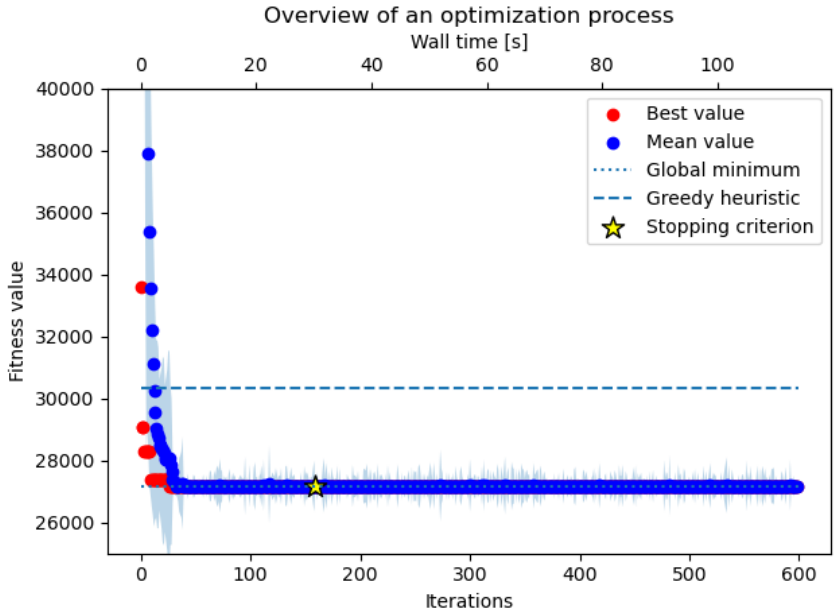
1. Python 3.8

2. Processor 11th Gen Intel(R) Core(TM) i7-1165G7 CPU @ 2.80GHz, 2803 Mhz, 4 Core(s), 8 Logical Processor(s)

5.2 tour29.csv

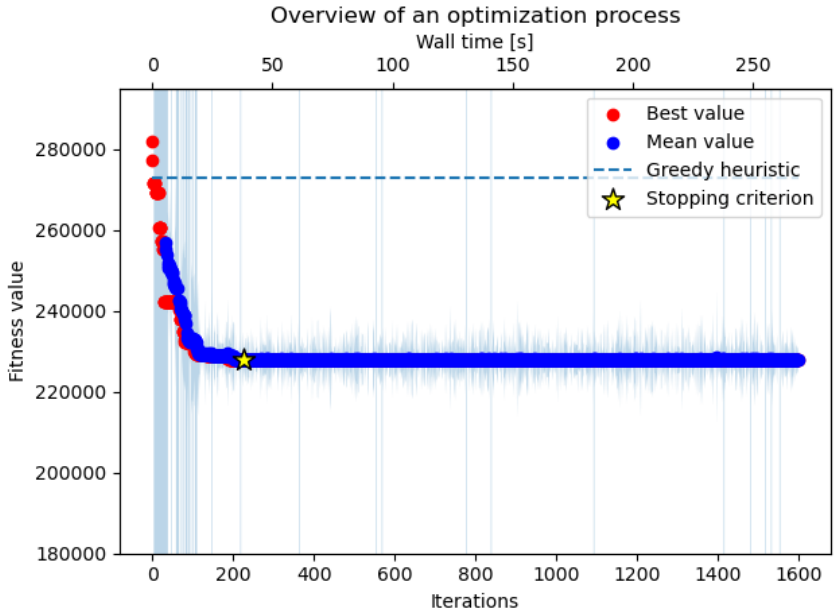
Setting the population to 600, the best path found was 5 0 1 4 7 3 2 6 8 12 13 15 23 24 26 19 25 27 28 22 21 20 16 17 18 14 11 10 9 with a tour length of 27,154 which is the optimal solution. The evolutionary algorithm was able to find fast the optimal solution after 15 seconds.

Due to time constraints, the evolutionary algorithm was not run 1000 times, but the 15 times it was run, it reached the optimal solution after 20 seconds in average, having a low standard deviation except for the spikes due to the mutation operator.



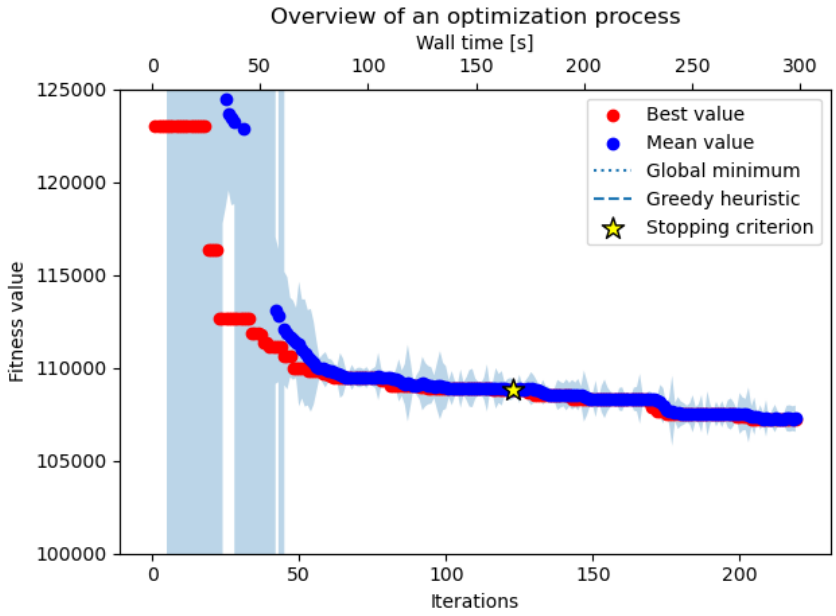
5.3 tour100.csv

Setting the population to 600, the tour length found was of 225,513 where it was found in 40 seconds. The evolutionary algorithm converged really fast but the quality of the solution is really good as it gave a solution with a tour length approx 40,000 less than the greedy heuristic. The diversity of the population is good at the start but diminishes quickly as it enters the convergence region with the occasional spikes due to the mutation operator.



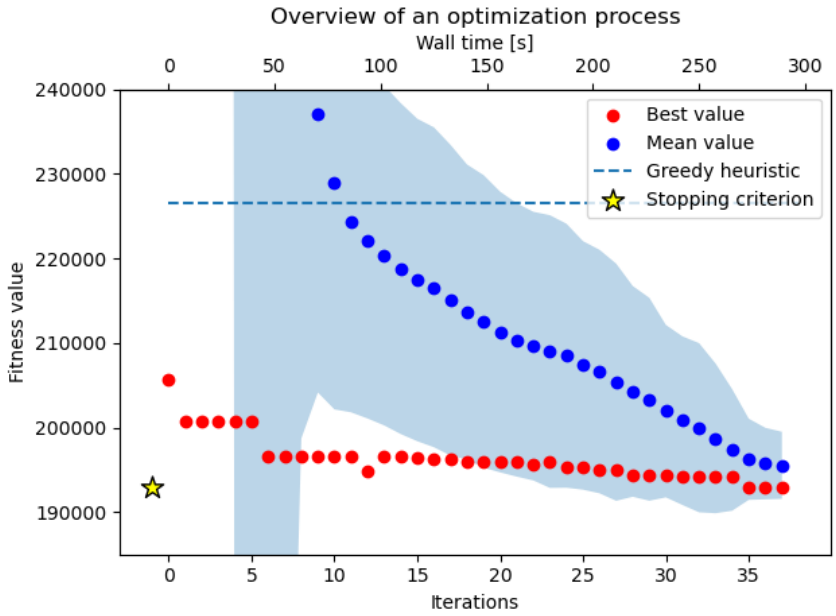
5.4 tour500.csv

Setting the population to 600, the tour length found was of 107,343 after the 5 min limit. The evolutionary algorithm converged fast after 50 iterations but the quality of the solution is not bad as it gave a solution with a tour length approx 15,000 less than the greedy heuristic. The diversity of the population is good at the start but diminishes quickly as it enters the convergence region with the occasional spikes due to the mutation operator.



5.5 tour1000.csv

Having a population of 500, the evolutionary algorithm found a good solution of 192,972, approx 35,000 better than the greedy heuristic. There was a good diversity of population at the start but by the end of the 5 min it had almost converged after almost 40 iterations, converging fast.



6 Critical reflection (target: 0.75 pages)

Three main strengths of evolutionary algorithms

1. The flexibility at which you can apply an an evolutionary algorithm to different problems

2. Parallelism can easily be applied to evolutionary algorithms
3. The ability to optimize the parameters with the evolutionary algorithm

Three main weak points of evolutionary algorithms

1. The number of parameters that need to be determined
2. The number of components that need to be chosen and tested several times.
3. Computational power required

Describe the main lessons learned from this project. Do you believe evolutionary algorithms are appropriate for the problem studied in this project? Why (not)? What surprised you and why? What did you learn from this project

Evolutionary algorithms are a useful way to solve np hard problems such as the traveling salesman problem. The concept of evolutionary algorithms is simple and may be applied broadly to optimization problems. I believe that a evolutionary algorithm is good to solve the TSP when the complexity is too high given the high number of cities but may not be the best way to solve the optimization when given a small number of cities due to the amount of parameters and components that need to be determined and implemented. What surprised me about this project is the amount of customization the evolutionary algorithms can have as there exist a high number of operators for each of the basic steps of an evolutionary algorithm, and the interaction of the exploratory vs the exploitation components of the algorithm.