Evolutionary Algorithms: Final report

Alexandros Kyrloglou (r0785071)

August 8, 2022

1 Metadata

- **Group members during group phase:** Andru Onciul and Alexios Konstantopoulos
- **Time spent on group phase:** 15 hours
- **Time spent on final code:** 35 hours
- **Time spent on final report:** 7 hours

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

1. Self adaptability of individual for: Mutation probability, Mutation magnitude, crossover probability, k-elimination and k-selection.
2. 1-opt Neighborhood search LSO (Local Search Operator) added, swapping 2 adjacent cities in a loop, in initialisation of the population and to the parents before the crossover.
3. Added heuristic, a path using a Nearest Neighbor algorithm is created and mutated, to initialise part of the population.

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

3.1 The three main features

1. Self-adaptability of individuals in every iteration for as many variables as possible
2. Addition of a LSO to converge faster with fairly low complexity of the LSO ($\mathcal{O}(n)$)
3. Jump start part of the population with simple heuristic initial path.

3.2 The main loop

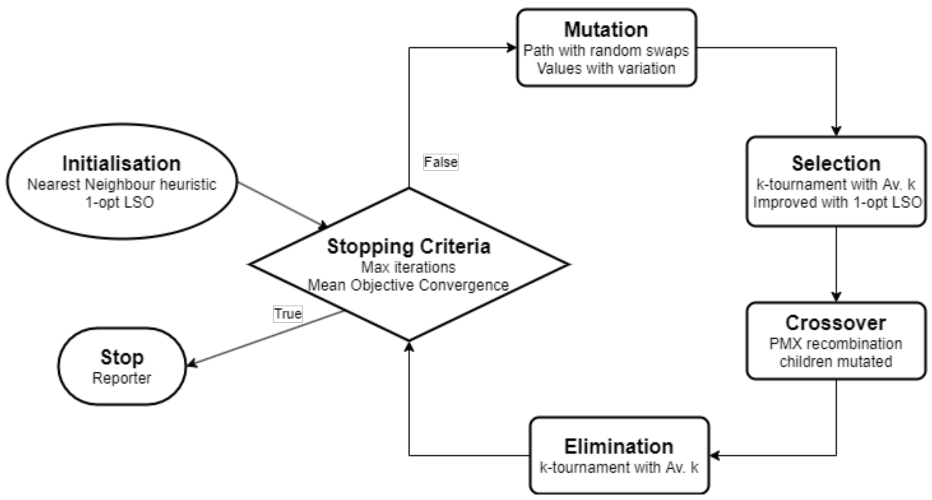

Figure 1: Main Loop of the Genetic Algorithm

3.3 Representation

Each candidate, Individual, is an object of the class r0785071.Individual. Each Individual has:

```
self.numberOfSwitches # The number of cities to be switched when mutating
self.mutationProbability # The probability of this Individual to get mutated
self.crossoverProbability # The probability of this Individual to be used as a parent
self.cost # The cost of the path of this Individual
self.kelimination # The k this Individual proposes for the elimination stage
self.kselection # The k this Individual proposes for the selection stage
self.path # The path of the Individual starting/ending from city 0 which is omited
self.numberOfCities # The number of cities in the problem
```

The most interesting part in for the representation is the way the path is saved. It is a $np.array$ of int. There are two optimisations in this representation. First by omitting the 0th city a bit of space in memory is saved and one more iteration in any loop that would go through the $numberOfCities$. Second, by setting the representation to always start from the 0th city all equivalent paths are eliminated. As the path is cyclic the same path could be calculated represented as different with another starting city. By fixing the starting point the searched space is reduced without loosing representational power.

When a new Individual is initialised the value $variation$ is also given. This value sets the percentage of random change for each parameter of the individual, this is done for self-adaptivity.

Two more values are given to all individuals, the $mutationVariation$ and the $kVariation$. The first is the percentage of change the parameters of an Individual get changed when it is mutated. And the second is by how much the parameter of the k-selection and k-elimination can vary initially or when mutated.

3.4 Initialization

For initialisation the number of Individual is found by trial and error, it must be big enough for sufficient diversity but low enough to keep the run time low.

At start all the individuals where initialised at random, with a random permutation of the number of Cities in the problem (without the 0th city as discussed before).

Two advanced initialisation mechanisms are implemented in order to speed the search of the space. A heuristic solution and a LSO. A heuristic solution is calculated using a Nearest Neighbour algorithm, this solution is mutated and used to set a percentage of the population, if this percentage is kept low then this solution should not take over the population. The second, LSO, is a search in the neighbourhood of every initial solution to start reducing the path cost even if slightly. The specifics of this LSO will be discussed in the subsection below 3.9.

These two mechanisms work best for smaller datasets, number of cities. Especially the LSO when used for very large problems takes too much time for the benefit it provides. Its complexity for the initialisation is $\mathcal{O}(n^2)$ as it checks all neighbours and calculates their path cost. For big datasets it was found better to turn them off or further optimisation of them is needed.

3.5 Selection operators

The ranking selection operator was initially considered, as it provides more control over the selective pressure. Ultimately it was not chosen as it is very computationally expensive. In order for a selection to be made every individual must have its fitness calculated and the whole population must be sorted.

The k-Tournament selection was chosen as the Selection operator because it has both reasonable selection power, meaning it chooses the best candidates, while keeping quite some variation to the population in order to search a big part of the space. The value of k is important as if set too high, only the best solutions will be picked and the algorithm will converge without searching the whole space, but if set too small then the selection is basically just randomly picking candidates. This value is initially set low in order to not reduce diversity to fast, but then let to self adapt through the iterations. Each Individual has a k value and their mean is calculated at the start of every iteration as the value for the k-selection tournament.

The selection is used to choose the parents for the crossover. After the parents are chosen they are improved slightly using the early stopping version of the LSO. In this way it is more likely that the children are converging to lower cost paths.

3.6 Mutation operators

When an individual is mutated there are two parts that get changed, its path and the parameters it has ($numberOfSwitches$, $mutationProbability$, $crossoverProbability$, $kelimination$ and $kselection$)

Firstly, for the path, the scramble mutation was considered as it searches a big part of the population space.

Quickly it was discovered that this mutation was mutating too much and as such information in good individuals seemed to be lost, making the algorithm not converge. Since the algorithm is trying to find a minimal path, the mutation chosen was swap mutation. In this way good candidates keep most of their information without being mutated much and the algorithm is sufficiently spanning the search space in order to find good results. Not all members of the population are mutated as there is a chance of mutation randomly mutating a part of the data-set. A parameter $percentageOfSwitches$ is initialised, that is used to calculate the number of switches per mutation (by multiplying this parameter with $numberOfCities$). A big value will increase the exploration but a too big value will change too much about the good individuals. Secondly, for the parameters, they get varied according to the $mutationVariation$ this is a hyper parameter set for all individuals for the value of the k-tourmanets another value is used this is the $kVariation$. By intrducing this variation the population is able to self-adapt these values.

3.7 Recombination operators

For the recombination a Partially Mapped Crossover (PMX) is used as seen in figure2. This was kept the same from the group stage as it seemed to give good results and the feedback we got from the other teams on this was positive. PMX creates off-springs that are half of each parent, in that way we keep most of the good information from both parents but we keep exploring because the half of each parent that is kept is chosen randomly by choosing the crossover range randomly. If the two parents have similar sub-paths, it is highly likely that the off-spring will have the same sub-path. There are no parameters for PMX so it is not possible to do self-adaptivity.

When a new offspring is created the average of the parameters of parents are used as its parameters. After each offspring is created it is given the chance to mutate to keep the population diversity high.

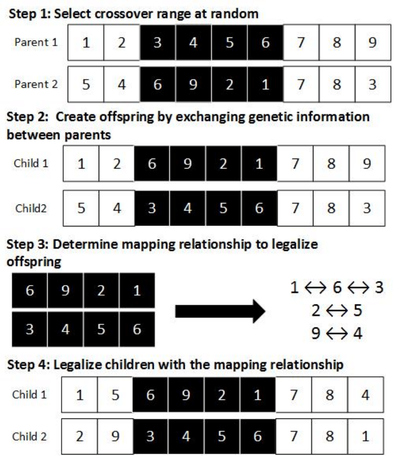
Step 1: Select crossover range at random
Parent 1
1 2 3 4 5 6 7 8 9
Parent 2
5 4 6 9 2 1 7 8 3
Step 2: Create offspring by exchanging genetic information between parents
Child 1
1 2 6 9 2 1 7 8 9
Child2
5 4 3 4 5 6 7 8 3
Step 3: Determine mapping relationship to legalize offspring
6 9 2 1
3 4 5 6
1 ↔ 6 ↔ 3
2 ↔ 5
9 ↔ 4
Step 4: Legalize children with the mapping relationship
Child 1
1 5 6 9 2 1 7 8 4
Child 2
2 9 3 4 5 6 7 8 1

Figure 2: Partially mapped crossover (PMX)
https://www.researchgate.net/figure/Example-of-partially-mapped-crossover_fig1_312336654

3.8 Elimination operators

The elimination chosen is a k-tournament elimination. The value of the k is the average value of all the Individuals. This value is left to self adapt through the iterations. The population given to this elimination is the concatenation of the old population and all the children created in this iteration.

3.9 Local search operators

One LSO is implemented with two different ways to be used. The LSO is a 1-opt neighborhood search. A neighbour is considered a path that has 2 adjacent cities swapped. The algorithm switches cities in random order this has complexity $\mathcal{O}(n^2)$ as it has to switch all cities and also calculate the cost of the path each time. There is an early stopping version set that stops when one better path is found as such this has complexity approximately equal to $\mathcal{O}(n)$ making it more usable in every iteration. The first version is used in the initialisation and the second is used to improve parents before the children are created.

Before introducing this LSO the population would not converge so easily and would vary for way to long close to good solutions without converging to them this helps to converge much faster.

3.10 Diversity promotion mechanisms

No specific diversity promotion mechanism is introduced, instead the diversitly is kept high with a high mutations.

3.11 Stopping criterion

There are two stopping criteria combined, a maximum number of iterations to make sure the algorithm stops and a way to see the convergence of the mean objective value. The second is the more impactfull part of the stopping criterion to initialise we use combination of the parameters $genForConvergence$ and $stoppingConvergenceSlope$. Here the $meanFitness$ for the past number of generations, defined as $genForConvergence$, is stored. This array is used in a linear regression and the slope of that line is stored. This slope shows how much the $meanFitness$ has changed in the past generations. Normalising this slope is also necessary, as the number of cities and the size of the distances will influence the mean value of the path's cost, as such the value of the mean fitness and its rate of change. By comparing it with the initialised $stoppingConvergenceSlope$ used as a minimum allowed value, initialised in the beginning, the stopping criterion is fully defined.

3.12 Parameter selection

The parameter selection was done mainly using intuition and trial and error. A lot of the parameters adapt during the execution with self-adaptation so at the initialisation they are set to make the population as diverse as possible and then they converge together with the population. This seemed to work quite well but mostly for smaller dataset as the algorithm is not efficient enough to run for a lot of iterations in the bigger datasets. As such the parameters are mostly set to work for the smaller datasets

4 Numerical experiments (target: 1.5 pages)

4.1 Metadata

```
populationSize = 200

useNN = True
percentageNN = 0.2
writeCSV = False
usedLSO_initialisation = True
useLSO_parents = True

kselection_init = 2
kelimination_init = 2
mutationProbability_init = 0.3
percentageOfSwitches_init = 0.1
crossoverProbability_init = 1
variation = 0.4

iterations = 1000
genForConvergence = 5
stoppingConvergenceSlope = 0.000001

printEveryIter = False
    class Individual:
        mutationVariation = 0.3
        kVariation = 2
```

Setup:

1. CPU: AMD Ryzen 9 5900HS, 3.30GHz (Base)
2. RAM: 16.GM at 4266MHz
3. Windows 10
4. Python: 3.9.12

4.2 tour29.csv

The test for 1000 times was not done but after running the problem quite a few times the result gotten varied from 160000-310000. In the figure 3 a run that found a good solution can be seen. This was one of the longest runs with usually taking less than 100 epochs to converge The algorithm does not get the optimal solution (probably). I think that more diversity in the search is needed to get better results, in order to explore the search space more. On average the algorithm stops in less than 20 seconds
best path: 7 13 14 18 12 6 25 11 20 21 22 26 17 28 10 23 1 19 27 4 3 16 2 24 15 9 5 8 0
best length: 164457.83 (32 Iterations)

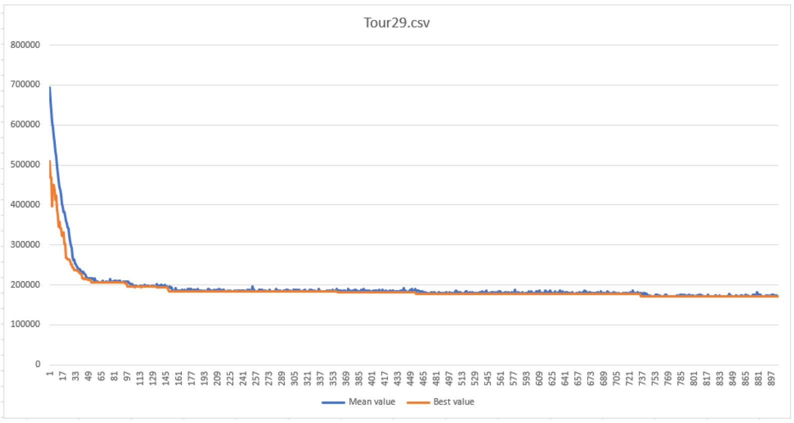

Figure 3: Example of results for tour29

4.3 tour100.csv

The algorithm does not get the optimal solution. I think that more diversity in the search is needed to get better results, in order to explore the search space more. But at least it converges to a quite fast. For most of the runs the algorithm stops in less than 100 iterations and in under 1 minute,
From the figure 4 the mean and the results from the first 2 iterations are removed as they either contain infinite or a have very big values making the plot unreadable.

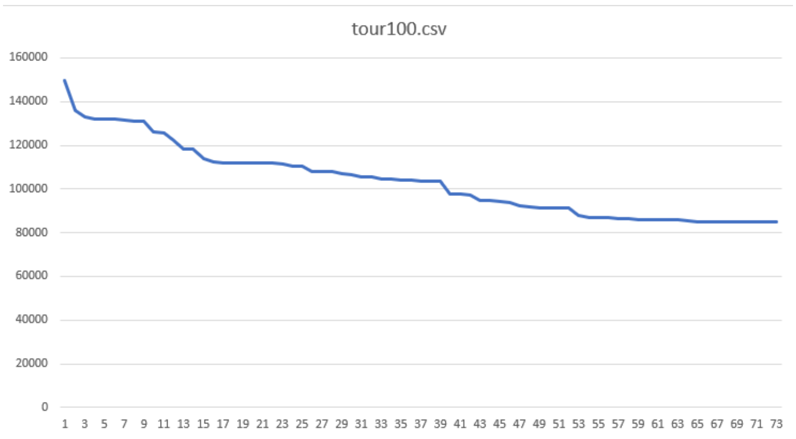

Figure 4: Example of results for tour100

best path: 74 5 66 47 77 27 79 81 93 69 42 62 87 34 51 78 75 50 16 63 65 19 88 76 23 45 64 70 96 85 55 25 36 20 82 38 80 30 1 58 17 29 60 21 83 43 92 7 3 2 53 13 54 18 57 95 89 73 15 49 8 84 4 41 31 56 28 10 22 35 44 72 9 67 86 11 32 12 61 24 91 33 52 46 39 40 71 6 37 59 90 94 48 97 68 26 14 99 98 0
best length: 84791.792 (74 Iterations)

4.4 tour500.csv

The algorithm does not get the optimal solution. I think that more diversity in the search is needed to get better results, in order to explore the search space more. But at least it converges to a quite fast. For most of the runs the algorithm stops in less than 20 iterations and in under 2.5 minutes.
From the figure 5 the first mean result is equal to ”inf”.
best length: 1200490.2625 (16 Iterations) a ”good” path was randomly found and population converged to it.

4.5 tour1000.csv

The algorithm runs out of time, more optimisations are necessary for it to be able to work with larger datasets an example run can be seen in figure 6. Multi-threading could be used in order to do this or the functions can be optimised in order to have lower complexity time. A major problem is probably memory management. The first couple of iterations are able to be computed quite fast and at some iteration each iteration starts to take more time to finish this can be seen in figure 7.

best length: 5018693.894 (12 Iterations)

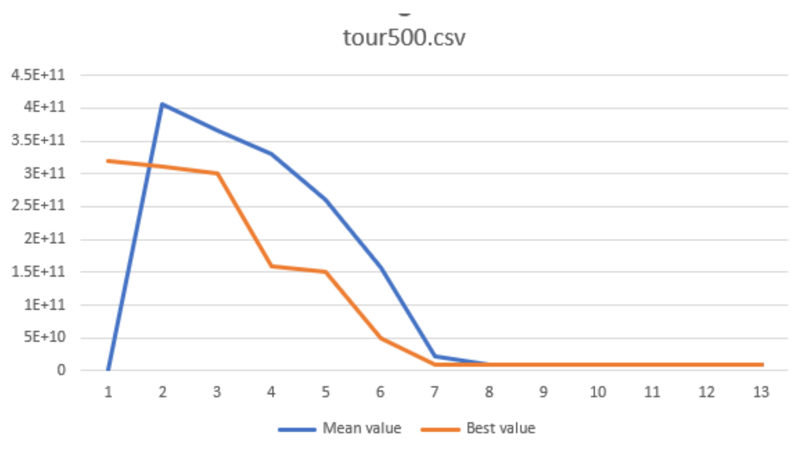

Figure 5: Example of results for tour500

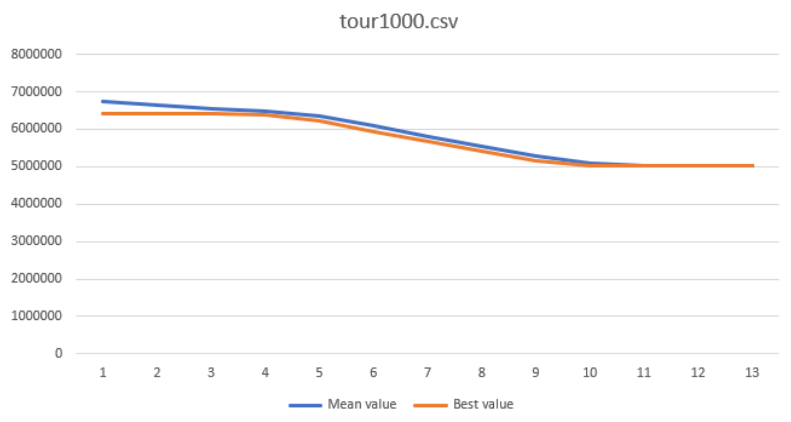

Figure 6: Example of results for tour1000

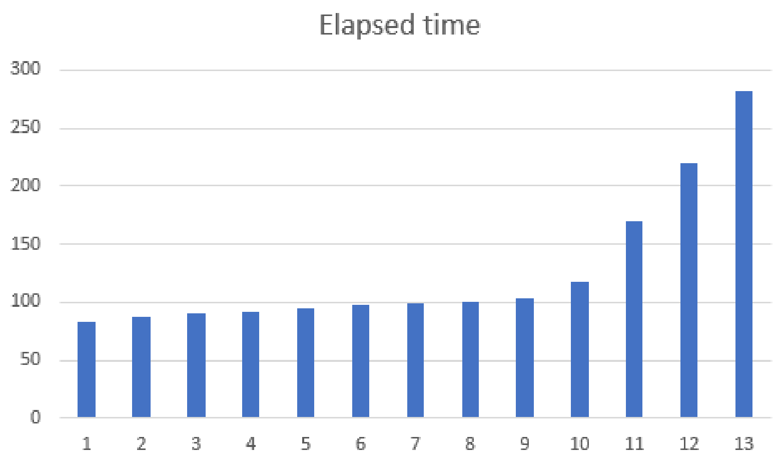

Figure 7: Example of Time per iteration tour1000

5 Critical reflection (target: 0.75 pages)

Main Advantages:

1. Can perform very well in various very different problems, and even in noisy situations.
2. It can balance how much exploration and exploitation is used in order to get good results at a reasonable time-span
3. Can take advantage of a lot of computational optimisations.

Main Disadvantages:

1. Takes a lot of knowledge and tweaking to get good results from an GA algorithm.
2. For problems with very high complexity a lot of computational resources are necessary in order to get results in good time.
3. They are slower than a lot of algorithms that find local minima than can be good enough in a lot of situations.

Genetic algorithms can be a very powerful tool for a lot of problems but take a lot to implement correctly and efficiently. For problems like the one in this project they are probably a very good choice. Maybe some combination with some AI techniques would be nice to investigate as well in order to see how they can be combined. In the end I found very interesting how hard it was to keep the population diverse and how many options and different implementations for each part of a GA exist.