Evolutionary Algorithms: Final report

Stijn Staring (r0620003)

December 8, 2020

1 Metadata

- **Group members during group phase:** Rajat Sharma and Pieter-Jan Vrielynck

- **Time spent on group phase:** 13 hours
- **Time spent on final code:** 72 hours
- **Time spent on final report:** 12 hours

2 Modifications since the group phase

2.1 Main improvements

1 Inver-over operator The final implementation of the evolutionary algorithm makes use of an inver-over operator. This operator tries to combine the advantage of unary operators as mutation and binary operators as cross over which has respectively low calculation cost and a notion of the population. This operator replaces the sequential constructive cross over and the scramble mutation operators. "SCX" is a suitable operator but not efficient enough for big problems and scramble is a naive method that breaks a lot of edges and can therefore damage good solutions. They where both replaced by inversion operators.

2 Enrichment of the initial population The "Nearest Neighbours" heuristic is used to initialize the population. In order to enhance the diversity it is not allowed to start multiple times from the same city when building up a solution. Afterwards the local search technique "3-opt" is applied for a couple of iterations. This local search technique is also used after the variation operators. After initialization of the population a best solution of 28074 is attained in 0.69 seconds which is better than the mean best solution during the group phase which was around 32250. For this result population size was set to 100.

3 Parallel selection and elimination After retrieving a good initial population, each individual is sequentially chosen to be adjusted by the means of variation operators. After a variable amount of adaptations the offspring is compared with its parent and the most fittest solution is included in the next generation.

2.2 Issues resolved

Slow algorithm due to high calculation load: When applying the algorithm of the group phase to the bigger problems e.g. tour 194, the algorithm became very slow. Solutions attained were therefore of poor quality. The calculation load is still high due to the vast size of the TSP problems, but it is mitigated by the use of the cheap inver-over operator (see Section 2.1). Also, the optimal population size is chosen to make an optimal trade off between calculation load and quality of the solution.

Worse results than simple greedy heuristic: Because of the use of a rich initial population (Section 2.1) the genetic algorithm already starts from a solution better than the simple greedy heuristic. This means that the gap that it has to bridge to reach the optimal solution is made much smaller.

Instantaneous loss of diversity: During the group phase good solutions rapidly took over the population which caused the algorithm to get stuck early in a local minima. This could be seen because the mean cost and the best cost quickly converged to each other which indicates that the diversity was completely gone. Therefore, the algorithm was not able to reach the optimal solution. The mean value of the best solution was still around a cost of 4000 away from the optimal solution after 10 runs on the smallest problem. Additionally, there was a lot of variation on the solutions. This follows from the poor domain exploration, whereby the local minima that were reached are mainly based on random factors during the initialization of the population, the selection operator

and the mutation operator. The reason for a rapid loss of diversity was a too high exploitation pressure that was introduced by making use of the λ/μ - elimination scheme in combination with a too high k value during k-tournament selection. Also, was initialized by introducing random routes. This not necessarily lead to a diverse population.
With the implementation of a parallel selection and elimination operator good solutions taking over the population is avoided.

Increasing cost of the best solution: During testing in the group phase it was found that the best solution sometimes increased in cost. Because a λ/μ - elimination scheme was used the reason for this will have been the mutation operator. By making use of the parent vs offspring comparison in the current implementation this behaviour is avoided.

3 Final design of the evolutionary algorithm:

3.1 Representation

Each individual solution is represented by an object of the Travelling Sales Man class. This structure allows for grouping different features i.e. cost of the route and the route followed itself. The order of the cities visited matches with the order of numbers that are listed and wherefore each number identifies a city. Lists are easy to modify and a lot of build in operators can be used. Also the list can intuitive be converted to a graph notation by assigning an edge from the current city to city on the next index. During cost calculation, the cost of the edge between the last and the first element should not be forgotten.

3.2 Initialization

How do you initialize the population? How did you determine the number of individuals? Did you implement advanced initialization mechanisms (local search operators, heuristic solutions)? If so, describe them. Do you believe your approach maintains sufficient diversity? How do you ensure that your population enrichment scheme does not immediately take over the population? Did you implement other initialization schemes that did not make it to the final version? Why did you discard them? How did you determine the population size?

The population is initialized in two steps. The first step comprises a "Nearest Neighbours" heuristic and the second step applies a "3-opt" local search operator(see Section 3.7).

The NN heuristic is implemented by choosing a random start city and always keep travelling from the current city to the nearest, next city. Additionally, to get a higher amount of diversity at a low calculation cost, it is expected to use every city only once to start from. It can be that the demanded population size is bigger than the amount of available start cities. In this case, after all the start cities have been used by the NN heuristic, randomly generated routes are inputted to the "3-opt" local search operator. The amount of iterations applied by "3-opt" is a parameter that is identified during the hyperparameter optimization. The time needed to initialize 100 individuals in the 29 tour problem takes around 0.7 seconds.

For bigger problems a rather small population is chosen to cut the calculation load. For the tour 194 in example, the initial population size is set to ****.

Rather small population to cut the calculation cost. For the small problem

Experimented with using NN for only half of the population and the rest randomly generated. Whereafter 3 opt was used. This only slowed down convergence, so used NN instead for the whole population. To improve diversity in the initial population, the NN is starting from a random start city that only can be selected once.

Because the use of the heuristic and local search –¿ started with a richer population and genetic algorithm was expected to do the small, last improvement until optimum.

3.3 Selection operators

Which selection operators did you implement? If they are not from the slides, describe them. Can you motivate why you chose this one? Are there parameters that need to be chosen? Did you use an advanced scheme to vary these parameters throughout the iterations? Did you try other selection operators not included in the final version? Why did you discard them?

Made use of competition based selection mechanism. The k tournament operator is chosen because the k value makes it easy to trade off exploration and exploitation and the operator can be easy and cheap implemented. The k -parameters are varied during the hypersearch.

Have chosen for k - tournament selection. A small adjustment with respect to the ones in the slides is the use of a variable k value. Start with small k –¿ more exploration . Later higher k, more exploitation. If more

individuals are selected, there is a higher change that among the selected individuals also a very good solution is chosen. This solution will rule out the rest which leads to more the same individuals used in the recombination operator and thus more exploitation towards this good solution.

K-tournament (keeping best individual but still some randomness),

Other operators tried are the exponentially decaying ranking with a decreasing s valu e over time. Results were similar.

In the final version selected every individual in the population for a variable amount of cross overs and mutations.

3.4 Mutation operators

Which mutation operators did you implement? If they are not from the slides, describe them. How do you choose among several mutation operators? Do you believe it will introduce sufficient randomness? Can that be controlled with parameters? Do you use self-adaptivity? Do you use any other advanced parameter control mechanisms (e.g., variable across iterations)? Did you try other mutation operators not included in the final version? Why did you discard them?

A mutation operator functions as a random local search and introduces randomly new edges in the population. It is often applied with a small probability in order not to damage good solutions to hard. Mutation is an unary operator which means that it only takes one individual as input and the operator is not population driven. The mutation that is finally chosen is a simple inversion mutation. Subset selected and inversed. This can be efficiently implemented –¿ only two edges change.

Can try a decreasing probability of mutation with the increase of the amount of generations. You don't want the mutation operator to ruin your solutions. And tried self-adaptivity which gave similar values as during the hyper search.

Also tried greedy mutation. Removing between 4 and 7 edges –¿ greedy reconnect the individual parts. Too expensive.

3.In the beginning generations it is good to have a large probability of mutation but towards the end it needs to be smaller

3.5 Recombination operators

Which recombination operators did you implement? If they are not from the slides, describe them. How do you choose among several recombination operators? Why did you choose these ones specifically? Explain how you believe that these operators can produce offspring that combine the best features from their parents. How does your operator behave if there is little overlap between the parents? Can your recombination be controlled with parameters; what behavior do they change? Do you use self-adaptivity? Do you use any other advanced parameter control mechanisms (e.g., variable across iterations)? Did you try other recombination operators not included in the final version? Why did you discard them? Did you consider recombination with arity strictly greater than 2?

Recombination is generally a binary operator with as goal to identify the good characteristics of the candidate solution and transfer this to the offspring. For the TSP problem it is therefore important to look at the overlap of two solutions and copy this to the offspring.

In the group phase had an okay implementation of a sequential constructive cross over operator. But even this too expensive. Overlap was copied to the offspring and for the other edges there was looked at the best alternative given by the two parents. This second step could however by extended as is done in the "Greedy recombination". Here the selection of the best alternative is not reduced to the domain of two parents but uses the best alternative given the whole population. The workflow of the "Greedy recombination" goes as follows. After the overlap between the parents is copied to the offspring and the rest of the edges are deleted, a greedy approach is used to make the offspring legitimate.

Overlap is copied to the offspring and the rest of the edges are deleted. A random start city is chosen and the offspring is gradually build up make use of a greedy approach.

Implemented the inver-over operator. Its power was its simpleness because the cross over was based on simple inversion of a sub part of an individual. This operator combines the advantage of unary and binary operators. This means a low cost and still a notion of the population in the selection of the subset to inverese. and thus was a cheap unary operator. The selection however of the segement was still population driven. Optimal solutions were consistently found for the tour29 problem.

inver - over –¿ selects city to inverse. The connection original in the parent is now embedded in the offspring.

downside –¿ inversion doesn't take the assymmetric cost matrix into account. This means that during an inversion a lot of edges are changed. Therefore there has be further looked at a greedy recombination operator. but too slow. Inversion combined with the more expensive 3 opt local search operator that is suitable for a ATSP problem, gave the best results.

3.6 Elimination operators

Which elimination operators did you implement? If they are not from the slides, describe them. Why did you select this one? Are there parameters that need to be chosen? Did you use an advanced scheme to vary these parameters throughout the iterations? Did you try other elimination operators not included in the final version? Why did you discard them?

The offspring goes through a variable amount of mutations and recombinations whereafter their fitness with the parent solution, original in the population, is compared. Hybrid method of age base elimination.

By definition the best solution will always remain which means that the up and down movements of the best solution that were present during the group phase are avoided.

A form of elitism is in effect.

Tried lambda + mu –¿ too exploitive. Tried k-tournament elimination

Would have been better if would check the similarity of edges between different solutions and based on this increase the cost and so decrease the probability that an individual would be selected. The implementation that was made was however to calculation expensive to be efficient. Therefore the cheaper method of preventing the same route to be included in the survivors has been used.

k tournament was implemented with the condition that the same individuals are discarded from the population when transferred to the survivors. It is thus not possible to select it, which ensures that the population in the next generation will all be different solutions of the problem.

Replaced by a diverse k tournament scheme. Element is removed from pop –¿ others more change to be selected. More randomness in the elimination and reduces the exploitation pressure and increases the exploration.

Experimented in elimination step with diversity mechanisms. (overlap edges/ same routes)

3.7 Local search operators

What local search operators did you implement? Describe them. Did they cause a significant improvement in the performance of your algorithm? Why (not)? Did you consider other local search operators that did not make the cut? Why did you discard them? Are there parameters that need to be determined in your operator? Do you use an advanced scheme to determine them (e.g., adaptive or self-adaptive)?

Naive three opt by swapping three randomly chosen edges on look at eight ways to reconnect them. If there was an improvement the swap was executed. A lot of zeros before found an improvement.

And efficient three opt based on two local searches that only considers the most potential cities for the three swap. Based on the condition that the new edges should make an improvement with respect to the old ones.

Add two figures of the two cases that are compared. Looking for a subsegment of the tour to insert between two nodes and optionally on top of that inverse.

Could let the local search run until 3 opt optimality for the tour 29 problem which enhanced convergence speed. However this became too expensive for the other problems. Therefore, in the current implementation only one loop over the whole route is considered.

Implemented a 3 opt local search operator and not 2 opt. 2 opt is inversion, 3 opt is insertion of a subpath and is enhanced with two local searches.

Graph of the flow of the algorithm –¿ one heuristic and local search technique.

Local search –¿ 3 opt (high calculation load) see section 4.7

3.8 Diversity promotion mechanisms

Did you implement a diversity promotion scheme? If yes, which one? If no, why not? Describe the mechanism you implemented. In what sense does the mechanism improve the performance of your evolutionary algorithm? Are there parameters that need to be determined? Did you use an advanced scheme to determine them?

In the current implementation diversity is ensured during the initialization of the population by always staring from a different start city as is discussed in section ??. Because only offspring and parents are compared, a good solution can't take over the population. Each candidate solutions is improving in parallel and benefits

indirectly through recombination by the increasing quality of the overall population.

clearly say what else have tried that it is not implemented in the final solution!!

Say that added diversity promotion in the elimination, because than you have a direct effect. This is not the case when you add a diversity mechanism in the selection step, because cross over and mutation can influence the diversity of the population.

3.9 Stopping criterion

Which stopping criterion did you implement? Did you combine several criteria?

Say that first used the difference between the mean of the population and the best but is not good with diversity condition. What used as stopping criteria –¿ of the best solution stayed constant for 80 generations in the tour 29 problem.

3.10 The main loop

Describe the main loop of your evolutionary algorithm using a clear picture (preferred) or high-level pseudocode. In what order do you apply the various operators? Why that order? If you are using several selection, mutation, recombination, elimination, and local search operators, describe how you choose among the possibilities. Are you selecting/eliminating all individuals in parallel, or one by one? With or without replacement?

In the final version applied a for loop over the population. While an individual is selected a variable amount of cross overs and mutations are applied.

Because of the for loop to select the individuals and as will further explained in Section **??**, there is only competition between a parent with its offspring.

3.11 Parameter selection

For all of the parameters that are not automatically determined by adaptivity or self-adaptivity (as you have described above), describe how you determined them. Did you perform a hyperparameter search? How did you do this? How did you determine these parameters would be valid both for small and large problem instances?

A hyper search is run with theses parameters: ...

These are the parameters that can vary: ...

Here is a table with the final parameters: ...

The parameters for the bigger problem can be obtained in a similar way.

Robust because only three parameters.

Self-adaptivity was tried for the mutation parameter. Similar results were attained as during the hyper search.

Amount of "3-opt" iterations performed.

3.12 Other considerations

Did you consider other items not listed above, such as elitism, multiobjective optimization strategies (e.g., island model, pareto front approximation), a parallel implementation, or other interesting computational optimizations (e.g. using advanced algorithms or data structures)? You can describe them here or add additional subsections as needed.

The operators are good, but solution can be improved towards cheaper implementations of these operators.

4 Numerical experiments

Goal: Based on this section and our execution of your code, we will evaluate the performance (time, quality of solutions) of your implementation and your ability to interpret and explain the results on benchmark problems.

To obtain the numerical tests described here, the implementations of the bigger problems are changed to the disadvantage of quality. Local search removed/less search. population made smaller.

4.1 Metadata

What parameters are there to choose in your evolutionary algorithm? Which fixed parameter values did you use for all experiments below? If some parameters are determined based on information from the problem instance (e.g., number of cities), also report their specific values for the problems below.

Report the main characteristics of the computer system on which you ran your evolutionary algorithm. In-

clude the processor or CPU (including the number of cores and clock speed), the amount of main memory, and the version of Python 3.

4.2 tour29.csv

Run your algorithm on this benchmark problem (with the 5 minute time limit from the Reporter). Include a typical convergence graph, by plotting the mean and best objective values in function of the time (for example based on the output of the Reporter class).

What is the best tour length you found? What is the corresponding sequence of cities?

Interpret your results. How do you rate the performance of your algorithm (time, memory, speed of convergence, diversity of population, quality of the best solution, etc)? Is your solution close to the optimal one?

Solve this problem 1000 times and record the results. Make a histogram of the final mean fitnessess and the final best fitnesses of the 1000 runs. Comment on this figure: is there a lot of variability in the results, what are the means and the standard deviations?

4.3 tour100.csv

Have to simplify the implementation.

Run your algorithm on this benchmark problem (with the 5 minute time limit from the Reporter). Include a typical convergence graph, by plotting the mean and best objective values in function of the time (for example based on the output of the Reporter class).

What is the best tour length you found in each case?

Interpret your results. How do you rate the performance of your algorithm (time, memory, speed of convergence, diversity of population, quality of the best solution, etc)? Is your solution close to the optimal one?

4.4 tour194.csv

Have to simplify the implementation.

Run your algorithm on this benchmark problem (with the 5 minute time limit from the Reporter). Include a typical convergence graph, by plotting the mean and best objective values in function of the time (for example based on the output of the Reporter class).

What is the best tour length you found?

Interpret your results. How do you rate the performance of your algorithm (time, memory, speed of convergence, diversity of population, quality of the best solution, etc)? Is your solution close to the optimal one?

4.5 tour929.csv

Have to simplify the implementation.

Run your algorithm on this benchmark problem (with the 5 minute time limit from the Reporter). Include a typical convergence graph, by plotting the mean and best objective values in function of the time (for example based on the output of the Reporter class).

What is the best tour length you found?

Interpret your results. How do you rate the performance of your algorithm (time, memory, speed of convergence, diversity of population, quality of the best solution, etc)? Is your solution close to the optimal one?

Did your algorithm converge before the time limit? How many iterations did you perform?

5 Critical reflection

Goal: Based on this section, we will evaluate your understanding and insight into the main strengths and weaknesses of your evolutionary algorithms.

Describe the main lessons learned from this project. What do you think are the main strong points of evolutionary algorithms in general? Did you apply these strengths in this project? What are the main weaknesses of evolutionary algorithms and of your implementation in particular? Do you think these can be avoided or mitigated? How? Do you believe evolutionary algorithms are appropriate for this problem? Why (not)? What surprised you and why? What did you learn from this project?

6 Other comments

In case you think there is something important to discuss that is not covered by the previous sections, you can do it here.