Evolutionary Algorithms: Final report

Jorrit Willaert (r0652971)

December 19, 2021

1 Formal requirements

The report is structured for fair and efficient grading of over 100 individual projects in the space of only a few days. Please respect the exact structure of this document. You are allowed to remove sections and . is the soul of wit: a good report will be **around** 7.5 **pages** long. The hard limit is 12 pages.

> Think of this report as a **take-home exam**; it will be used at the exam for structuring the discussion and questions. Make an effort so that it can be visually scanned efficiently, e.g., by using boldface or colors to highlight key points, using lists, clearly defined paragraphs, figures, etc.
>
> You do not need to explain in this report **how** the techniques and concepts that are literally in the slides work. The goal of this report is **not** to illustrate that you can reproduce the slides. You need to convince me that you aptly used these (and other) techniques in this project. If I have doubts about your understanding of certain concepts in the course materials, I will test this hypothesis at the exam.

It is recommended that you use this LaTeX template, but you are allowed to reproduce it with the same structure in a WYSIWYG-editor. The purple text containing our evaluation criteria can be removed. You should replace the blue text with your discussion.

This report should be uploaded to Toledo by December 31, 2021 at 16:00 CET. It must be in the **Portable Document Format** (pdf) and must be named `r0123456_final.pdf`, where r0123456 should be replaced with your student number.

2 Metadata

- **Group members during group phase:** Lukas De Greve and Thomas Vanhemel
- **Time spent on group phase:** 10 hours
- **Time spent on final code:** 40 hours
- **Time spent on final report:** 10 hours

3 Peer review reports (target: 1 page)

Goal: Based on this section, we will evaluate insofar as you are able to recognize and analyze common problems arising in the design and implementation of evolutionary algorithms and your ability to effectively solve them.

3.1 The weak points

List the (up to six distinct) weak points that were identified in the two peer review reports that you received. Use at most 3 sentences to describe each weak point.

1. Our initial recombination operator was a simplified version of the edge crossover operator [2]. However, this version did not prioritize common edges between parents, but only chose an entry which itself had the shortest list. Hence, not enough exploitation of the parents features follows.
2. The $(\kappa + \mu)$-elimination (without any type of diversity promotion) puts a lot of selective pressure on the population.
3. Our mutation operator does not scale to larger problems, since it only swaps two random locations. As a consequence, the mutation operator will have a relatively even smaller impact on the solution when the problem size increases.
4. Due to the elimination strategy, together with the chosen mutation operator, premature convergence was observed.

3.2 The solutions

List the solutions to the identified weak points. Use at most 3 sentences to describe your solution. You do not need to explain the techniques in detail if they are in the slides or handbook; just state something like "Weak point X was solved/mitigated/diminished by using the island model as diversity promotion scheme." Note there could be more or less solutions than the number of weak points.

1. The simplified version of the edge crossover operator was first replaced with the proper edge crossover operator [1]. However, once the running times of edge recombination were compared with the ones of order crossover, it was apparent that order crossover is much faster. For this reason, edge crossover was abandoned.
2. The $(\kappa + \mu)$-elimination was kept for quite some time, but was supplemented with fitness sharing. The high selective pressure of $(\kappa + \mu)$-elimination was largely mitigated With the introduction of this diversity promotion scheme. However, to further reduce the selective pressure present in the $(\kappa + \mu)$-elimination, the elimination eventually also used k-tournament (along with the same fitness sharing technique).
3. The mutation operator has been changed from swap mutation to inversion mutation. Hence, the effect of the mutation operator is constant for rising problem sizes.
4. With the introduction of fitness sharing, along with the inversion mutation operator, premature convergence was largely avoided.

Which weak points did you not address? List them and briefly motivate (target, 3 lines each).

1.
2.

3.3 The best suggestion

Among the two suggestions that you received and the two suggestions that you gave, which one did you find the most helpful? Briefly describe the suggestion and why you think it was the best. (suggested maximum: 10 lines)

Both groups suggested to modify the simplified edge crossover operator to the 'proper' one. Although this certainly is a useful suggestion, another suggestion by one group is even more useful in my opinion, given that I was going to change the crossover operator in the individual phase anyway. The other suggestion is about changing the mutation operator from the swap mutation to the inversion mutation. In the group phase part of the project, we hadn't really observed the shortcoming of the swap mutation, which became clear after their suggestion has been made.

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

List the main changes that you implemented since the group phase. You do not need to explain the employed techniques in detail; for this, you should refer to the appropriate subsection of section 3 of the report. Naturally, there can be overlap with the solutions from the previous section.

1. The simplified edge recombination operator was first replaced with the proper edge recombination operator, after which it eventually was replaced by the order recombination operator due to massive speed gains. An elaboration on the recombination operators is provided in Section 5.7).
2. Fitness sharing has been introduced in the elimination step, as further explained in Section 5.10.
3. A local search operator (2-opt) has been introduced to increase the fitnesses of the newly created offsprings (Section 5.9).
4. Simple random initialization has been replaced by greedy and legal initialization, as further explained in Section 5.4.
5. The mutation operator has been changed from the swap mutation to inversion mutation. This is more elaborated in Section 5.6.
6. The elimination operator has eventually been changed from $(\kappa + \mu)$-elimination to k-tournament elimination, for reasons explained further in Section 5.8.
7. Numerous optimizations have been made in almost all parts of the algorithm to massively increase the execution speed.

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

Goal: Based on this section, we will evaluate insofar as you are able to design and implement an advanced, effective evolutionary algorithm for solving a model problem.

5.1 The three main features

List the three main components of your evolutionary algorithm for this project. That is, what are its most distinctive characteristics, what components am I not allowed to change to a more basic version? Ideally these are some of the more advanced features that you added since the group phase.

1. Fitness sharing has been used in the elimination step of the algorithm. This diversity promotion scheme is of crucial importance to avoid premature convergence, and hence makes sure that far better solutions can be found, instead of letting all individuals converge to one local minima.
2. By introducing the 2-opt local search operator, far better solutions were quickly found. Without the local search operator, much more iterations were required to find the same fitness values, along with a necessary larger population. Although this operator is inherently extremely computationally expensive, it is turns out to be pivotal in the algorithm. Especially in this operator, optimizations such as making use of dynamic programming and using Numba were decisive in making the operator computationally feasible.
3. One last crucial improvement is the introduction of the greedy and legally initializations. Greedy initialization starts from a random node, and chooses the next one according to the smallest distance. The details of this initialization scheme are elaborated in Section 5.4, along with a consideration of the introduced biases. Furthermore, legal initialization simply chooses the next node from a random neighbor that has an existing road between them.

5.2 The main loop

5Make a picture of the "flow" in your evolutionary algorithm, similar to the example below. Include all the main components (mutation, recombination, selection, elimination, initialization, local search operators, diversity promotion mechanisms). There are no formal requirements on how to do this, as long as it is clear and you can efficiently explain your complete evolutionary algorithm using this picture at the exam. Contrary to the picture below, include the specific techniques, e.g., top-λ elimination, k-tournament selection, where possible.

The questions we ask from section 5.3 onwards in blue are there to guide which topics to discuss, rather than an exact list of questions that must be answered. Feel free to add more items to discuss.

5.3 Representation

Possible solutions are represented as **permutations** and are written down in **cycle notation**. E.g. the permutation (1423) starts at node 1, then goes to 4, then 2, then 3 and returns to 1. An advantage of this notation is that no cycles are present as long as we initialize the representations as a random permutation of the list of all nodes.

This representation is implemented in our program as a numpy array with length equal to the number of nodes in the problem. Each element in the array consists of one integer number: the node number.

5.4 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 non-symmetric distance matrix gets passed along as an argument to the initialization of an individual. In the group-phase part of the projects, individuals were generated by a random permutation, with their size determined from the distance matrix. However, especially for the larger problem size, quite a lot of paths were nonexisting. Hence, random initialization of all individuals yielded almost always individuals where non of them represented a valid path. Therefore, 10 % of the individuals were 'greedily' initialized. Here, a random city is chosen as the first one, after which the next city is chosen as the city with the smallest distance from the current one. This process repeats itself, until a whole valid tour has been established. If, however, suddenly all the distances to possible successive cities are nonexisting, then the initialization of that individual restarts from the beginning.

An individual also gets assigned a random α value, which represents the probability that the individual will mutate in the mutation step of the algorithm.

5.5 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? The k-tournament selection operator was implemented because exploration of suboptimal solutions is promoted. After several experiments, a k-value of was chosen.

5.6 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? The mutation operator used for the final implementation is the inversion mutation, whereby a random sub-vector is chosen and its order is reversed.

The mutation rate is specific to each individual, and initialized as ... with a random variation of ... . Furthermore, self-adaptivity is used ...

5.7 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?

In the group phase part of the project, a simplified version of the edge crossover was used as the recombination algorithm. This process is described in Algorithm 1 [2]. This recombination results in a new path where all edges of the child were present in at least one of the parents. It does however not prioritize edges present in both parents over edges present in a single parent.

Algorithm 1 Simple edge recombination operator

Let K be the empty list
let N be the first node of a random parent
while length(K) $<$ length(Parent) **do**
    Append K to N
    Remove N from all neighbour lists
    if N's neighbour list is not empty **then**
        let N^* be the neighbor of N with the fewest neighbours in its list (or a random one, should there be multiple)
    else
        let N^* be a randomly chosen node that is not in K
    end if
end while
N $\leftarrow N^*$

This algorithm is very simple and was the weakest part of the genetic algorithm. However, the algorithm still has some desirable features despite its simplicity. When N consecutive edges are present in both parents (N $\geq$ 2), the child will contain at least N/2 of these edges, so important features are mostly preserved. On the other hand, when the parents are very different, the child will look fairly different from both parents. This combined with some aspects of randomness in the algorithm makes it decent at exploring the different solutions.

The reason this simplified algorithm was implemented, instead of the proper one from Eiben & Smith [1], was due to the belief that the computational cost of this algorithm was (much) lower than the one from Eiben & Smith.

For the individual phase of the project, an analyses was made between order crossover and the proper edge crossover algorithm of Eiben & Smith. After some research, with a lot of contradictory advices, an arbitrary choice has been made to first try out the proper edge crossover algorithm.

Implementation wise, quite a lot of effort has been made to catch all the corner cases of the algorithm, along with achieving relatively optimized code. The algorithm was ket a long time thereafter, until it was noticed that for the large problem sizes, crossover took an extremely long time (up to 95 % of the total runtime was spend in the edge crossover operator).

Due to this slow execution time, order crossover [1] has been implemented as well.

This crossover algorithm is inherently much cheaper to calculate and takes only about 5 % of the total execution time in the final algorithm. This is exactly the reason why the crossover algorithm was eventually used.

5.8 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? K-tournament and (μ + λ)-elimination were considered. Ultimately, (μ + λ)-elimination was chosen, although it provides a lot of selective pressure.

5.9 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)?

The 2-opt local search operator has been implemented, which swaps every two possible edges in a given cycle. In a first version of this algorithm, the fitness was recalculated for every possible 'neighbor' of the given individual, which entailed an unacceptable high computational cost, especially for the larger problem sizes. After some investigation, patterns were detected in the computation of the fitness. Hence, instead of recalculating the fitness for every neighbor, some kind of dynamic programming approach was undertaken. For every individual, there is a sort of preprocessing step, whereby so-called 'cumulative' are created. These cumulatives capture the path length from the first city to that corresponding city in the cumulative array. The same process applies for the calculation of the path length from the last city to the corresponding city in the array (i.e. in reverse order, whereby the return cost of the last city to the first city is also incorporated). It is clear that the calculation of these cumulatives is done in $O(N)$, where N is the number of cities in the problem size.

Now, calculations of fitnesses of individuals are simply a matter of bookkeeping. The process is explained in Algorithm 2.

Algorithm 2 Local search operator

```
Let the best fitness be the fitness of the original individual
Let the best combination be (0, 0)
Build cumulatives
for first from 1 to (length - 3) do
    fit_first part = forward_cumulative[first − 1]
    fit_middle part = 0.0
    for second from (first + 2) to (length - 1) do
        fit_middle part += distanceMatrix[order[second − 1]][order[second − 2]]
        fit_last part = backward_cumulative[second]
        first_bridge = distanceMatrix[order[first − 1]][order[second − 1]]
        second_bridge = distanceMatrix[order[first]][order[second]]
        fitness = fit_first part + first_bridge + fit_middle part + second_bridge + fit_last part
        if fitness < best_fitness then
            Let the new best combination be (first, second)
            Let the new best fitness be the newly calculated fitness
        end if
    end for
end for
Swap the order of the individual from from the best first and best second, obtained from the best combination.
```

It should also be noted that by using Numba with the command $@jit(nopython = True)$ above the method declarations, the local search operator runs **745 times as fast**. Numba can make these huge improvements due to compilation of these methods, where especially the looping contributes massively to the speedup.

5.10 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?

5.11 Stopping criterion

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

5.12 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?

5.13 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.

6 Numerical experiments (target: 1.5 pages)

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.

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

6.2 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?

6.3 tour100.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 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?

6.4 tour500.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?

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?

6.5 tour1000.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?

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?

7 Critical reflection (target: 0.75 pages)

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

What are the three main strengths of evolutionary algorithms in your experience?

1.
2.
3.

What are the three main weak points of evolutionary algorithms in your experience?

1.
2.
3.

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?

8 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.

References

[1] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer International Publishing, 2 edition.

[2] Darrell Whitley, Timothy Starkweather, and Daniel Shaner. The Traveling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Recombination. page 18.