Evolutionary Algorithms: Project

Sahar Chehrazad
Nick Vannieuwenhoven

October 8, 2021

1 Formal requirements

This document describes the 2021–2022 project assignment for *Genetic Algorithms and Evolutionary Computing*. **Read this assignment carefully before starting and closely follow its instructions**.

- You will implement an evolutionary algorithm for the **traveling salesperson problem** in **Python** 3.9.
- The Python code that you submit will be executed automatically, so it is imperative to follow the exact specifications and interface set out here.
- Failure to adhere to the instructions in this assignment will result in a deduction of credit. *The choice of programming language is non-negotiable.*

The project is essentially a **take-home exam**:

- The intermediate report and peer reviews comprise 10% of the final grade.
- The final report and final Python code comprise 60% of the final grade.
- The discussion of your project along with general questions about evolutionary algorithms at the exam will contribute the final 30% of your score for this course.
- If you do not hand in one of the deliverables below, the exam score will be NA.

All of the following instances of "r0123456" should be replaced with your own student number.

There are **three deliverables** and likewise three deadlines to observe:

1. *November 12, 2021 at 16:00 CET*: The **intermediate report** of the group phase.
2. *November 19, 2021 at 16:00 CET*: The **peer review reports**.
3. *December 31, 2021 at 16:00 CET*: The final **Python code** and the **final report** of the individual phase.

These deliverables should be **uploaded to Toledo** by their respective deadlines. The reports must be in the **Portable Document Format**. The deliverables should be named as follows.

1. The reports must be named `r0123456_intermediate.pdf` and `r0123456_final.pdf` for the group phase report and final report respectively.
2. The peer reviews should be named `r0123456_peer1.pdf` and `r0123456_peer2.pdf`.
3. The program code must be a **single** Python source code file named `r0123456.py`.

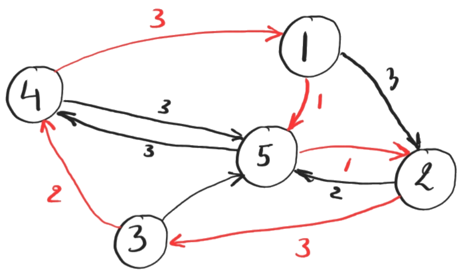
Templates of the deliverables that you must use are provided on Toledo. As the report templates will comprise a major portion of the score for this project, it is strongly advised to study the template carefully so you understand which aspects are critical in our assessment. The report templates also specify a minimum set of experiments that you must include and discuss. *The code template is discussed in section 4.*

The output of the Python code that you submit will be **graded automatically** by our testing framework. We will apply your code to five **benchmark problems**, similar to those that are posted on Toledo along with this assignment. For each benchmark problem, your code will be given a **fixed time budget** of 5 minutes. To ensure a level playing field, your code will be **restricted to utilize** 2 **physical cores**. The grade you receive for the code will be partly based on the performance of your best solution relative to your peers.

The goal of this project is that you design, implement, and analyze an evolutionary algorithm yourself, hence **you cannot use existing Python evolutionary algorithms frameworks**. When in doubt, contact the teaching assistant.

2 Problem statement

The **traveling salesperson problem** consists of minimizing the length of a cycle that visits all vertices in a weighted, directed graph. The length of a cycle is defined as the sum of the weights of all directed edges constituting the cycle. For example, the cycle defined by the red edges in the figure below has length 10:



Note that this picture includes vertices that are not directly connected by one another, such as 1 and 3. This corresponds to an edge weight of $e_{1,3}$ equal to ∞.

Formally, let (V, E) be a weighted, directed graph with n vertices V and directed edges $E \subset V \times V$. An edge from $v_i \in V$ to $v_j \in V$ is denoted by $e_{i,j} = (v_i, v_j)$. The weights, or distances, of the edges are given by a **nonsymmetric distance matrix** $D \in \mathbb{R}^{n \times n}$: $d_{i,j}$ represents the weight of $e_{i,j} = (v_i, v_j)$. Note that $d_{i,j}$ can be different from $d_{j,i}$. The length of the cycle (or cyclic permutation) $\pi = (\pi_1 \pi_2 \pi_3 \cdots \pi_n)$ is then defined as

$$\text{length}(\pi) := d_{\pi_n,\pi_1} + \sum_{i=1}^{n-1} d_{\pi_i,\pi_{i+1}}.$$

The goal of the traveling salesperson problem is to solve the optimization problem

$$\min_{\pi \in C_n(V)} \text{length}(\pi),$$

where $C_n(V)$ is the set of all $(n-1)!$ cyclic permutations on the n vertices V. The objective function that we will use to evaluate your project is the length of the cycle.

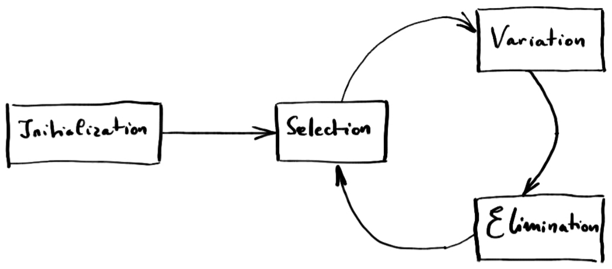
3 Work plan

The project consists of two phases: a **group phase** in groups of 3 that mainly takes place during four exercise sessions, and a more substantial **individual phase** that you complete individually. **Collaboration is allowed only during the group phase**, and only with your group members. The goal of the group phase is to have a minimal working evolutionary algorithm utilizing only the basic components (initialization, selection, variation, elimination) that you can extend, improve, and optimize in your individual phase.

3.1 Group phase (10h)

During the group phase you work in a group of 3 to design and implement a basic evolutionary algorithm. The composition of the groups will be communicated in the first exercise session. The following tasks are to be completed in this phase.

Exercise session 1: Think about all the components that you need for a basic evolutionary algorithm:



Discuss and decide which representation to use, what selection and elimination to perform, and what variation operators could be designed. Try to reason how these four components would interact with one another.

Exercise session 2: Implement a representation, the objective function length, a simple population initialization, a selection mechanism, one mutation operator, one recombination operator, an elimination mechanism, and a stopping criterion. **Start from the code template** that is provided on Toledo and further described in section 4.

Exercise session 3: Test your basic evolutionary algorithm on the smallest benchmark problem on Toledo. Identify and list all problems, bottlenecks, and inefficiencies with your current basic algorithm, but do not attempt to fix them (except for mere programming errors). **Write the group phase project report using the template.** Each group member should hand in this report on Toledo by November 12, 2021 at 16:00 CET. No code must be submitted for this phase.

Exercise session 4: You will receive the intermediate reports from two groups. Read these reports and discuss the evolutionary algorithms presented therein with your group. What are the strong and weak points? Do you think the components would interact well with one another? Can you think of solutions for their problems? Write down your conclusions in **two peer review reports**, one for each group, **using the template on Toledo**. Each group member should hand in both peer reviews on Toledo by November 19, 2021 at 16:00 CET. In the following week, you will receive two peer reviews about your own group report. You can use the insights from these peer reviews to improve your evolutionary algorithm in the individual phase.

3.2 Individual phase (40h)

In the individual phase you continue from the code that you prepared in the group phase with your group. From this point onwards, the project is an individual assignment. **Cooperation is strictly forbidden in this phase.**

The goal of this phase is that you develop an advanced, optimized evolutionary algorithm for the traveling salesperson problem. You may want to implement additional or more advanced selection schemes, elimination mechanisms, variation operators, local search operators, population enrichment schemes, parameter self-adaptivity, optimized hyperparameter search, diversity-promoting schemes, and so on. You should also write an individual final report using the final report template. The Python code and final report should be turned in via Toledo by December 31, 2021 at 16:00 CET.

4 Code template

The code template from which you start is as follows.

```
import Reporter
import numpy as np

# Modify the class name to match your student number.
class r0123456:

  def __init__(self):
    self.reporter = Reporter.Reporter(self.__class__.__name__)

  # The evolutionary algorithm's main loop
  def optimize(self, filename):
    # Read distance matrix from file.
    file = open(filename)
    distanceMatrix = np.loadtxt(file, delimiter=",")
    file.close()

    # Your code here.
    yourConvergenceTestsHere = True
    while( yourConvergenceTestsHere ):
      meanObjective = 0.0
      bestObjective = 0.0
      bestSolution = np.array([1,2,3,4,5])

      # Your code here.

      # Call the reporter with:
      #  - the mean objective function value of the population
      #  - the best objective function value of the population
      #  - a 1D numpy array in the cycle notation containing the best solution
      #    with city numbering starting from 0
      timeLeft = self.reporter.report(meanObjective, bestObjective, bestSolution)
      if timeLeft < 0:
        break

    # Your code here.
    return 0
```

You should **replace the class name** with your student number. You can add additional code to the constructor. The testing framework will call the default, no-argument constructor.

The class should have at least one method, `optimize`. The testing framework will call `optimize` with one argument, a string containing the filename of the traveling salesperson problem instance to optimize. You are allowed to add optional arguments to this method. The main loop of your evolutionary algorithm should be implemented by this method. The first three lines of this method should read the matrix, as in the template. The last three lines of the main evolutionary algorithm loop should report your results using the Reporter class provided on Toledo as in the template. The `float` returned by `Reporter.report` method indicates the amount of time in seconds that is left. These last three lines of the loop must not be modified. The stopping criterion `yourConvergenceTestsHere` can be implemented as you see fit.

Everything else you can implement as you desire (extra classes, methods, attributes, etc), subject to the constraint that you can use only 1 module (code file).