Evolutionary Computation

Group Assignment: The Traveling Salesperson Problem

Implementation + Video Report

Due date: 11:55pm, 27 September 2019 [this is 1 week after the last lecture]

1 Overview

Assignments should be done in groups consisting of five (5) students.

This assignment has to be handed in and reasonable effort has to be made to complete it in order to pass the course. The minimum requirement is that the different modules of evolutionary algorithms for the TSP need to be implemented as described below and that evolutionary algorithms for the TSP have to be designed and tested on the benchmarks given in TSPlib.

2 Assignment

The goal is to design a library and implement evolutionary algorithms for the traveling salesperson problem (TSP). The input for the TSP is given by n cities and distances d_{ij} for traveling from city i to city j, $1 \leq i, j \leq n$. A tour starts at one city, visits each city exactly once, and returns to the origin. The goal is to compute a tour of minimal cost. The TSP is one of the most famous NP-hard optimization problems and you should build an EA library and design evolutionary algorithms for this problem.

When designing your library for the TSP, it is desirable to follow object-orientated design practices and build a modular system. It should be possible to extend the system by using different methods for each part of the design, e.g., different individual representations, operators, and selection methods. In the following, we list the different modules that need to be implemented for the TSP problem. These are basic requirements and you may feel free to add additional features and operators to your library. If the parameter setting is not specified in detail then you are free regarding your choice. You should, however, take into account the recommendations given in the lecture.

You can use any of the following programming languages: Java, Python.

Exercise 1 *Problem Representation and TSPlib (5 points)*

Write a class *TSPProblem* which represents the TSP problem. Your class should enable the construction of TSP problems from the files of the symmetric traveling salesperson problem of the TSPlib, which is available online:

```
For the problem files:
http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/
The TSP main page has more information for you, if you want to read a bit:
http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
```

Exercise 2 *Individual and Population Representation (10 marks)*

Represent a possible solution to the TSP as a permutation of the given cities and implement it in a class *Individual.* Evolutionary algorithms often start with solutions that are constructed uniformly at random. Write a method that constructs such a solution in **linear time** – not in O(n log n) or even O(n^2), but in O(n), where n is the number of cities.

A population in an evolutionary algorithms represents a set of solutions. Implement a class *Population* for representing a population which is a set of individuals. Make sure that you can evaluate the quality of a solution with respect to a given problem.

Exercise 3 *Variation operators (24 marks)*

- Implement the different mutation operators (insert, swap, inversion, scramble) for permutations given in the lecture.
- Implement the different crossover operators (Order Crossover, PMX Crossover, Cycle Crossover, Edge Recombination) for permutations given in the lecture.

Exercise 4 *Selection (9 marks)*

Implement the different selection methods (fitness-proportional, tournament selection, elitism) given in the lecture.

Exercise 5 *Evolutionary Algorithms and Benchmarking (20 marks)*

- Design three different evolutionary algorithms using crossover and mutation, based on the implementation of your different modules.
- *(Experiment 1)* Run your three algorithms with population sizes $10, 20, 50, 100$ on the instances EIL51, EIL76, EIL101, ST70, KROA100, KROC100, KROD100, LIN105, PCB442, PR2392 from TSPlib. Report the outcomes after 5000, 10000, and 20000 generations. To clarify: these are 3*4*10=120 runs, i.e., 3 algorithms X 4 population sizes X 10 instances.
- *(Experiment 2)* From these three algorithms, run your best algorithm with a population size of 50 for 10000 generations on the ten TSPlib instances mentioned above. Report for each instance either (1) the average cost and the standard deviation or (2) the median cost and the interquartile range.

Notes:

- For the large instance, e.g., PR2392: if you cannot finish a single run, please report the results that you can achieve within some time limit, e.g., "after 4 hours" or "after 8 hours".
- How often you repeat the experiment involving your best algorithm is up to you. 10 repetitions is a good start, 30 or 100 repetitions will result in more reliable means/medians, however, you might not have the time to run the experiments that often.
- How to report the outcomes: for each of *Experiment 1* and *Experiment 2*, submit the results in a plain text file (no Excel spreadsheet or similar). This means that you will have to submit in total two text files for this Exercise.

It is absolutely critical that you provide detailed instructions on how to reproduce the results. With this, we mean that the TAs have to be able to run your code and get results that are somewhat similar to what you are reporting.

Exercise 6 *Video Report (32 marks)*

Create a short video (4-5 minutes) **in English** in which you

1. introduce the team;
2. describe the three different algorithms you have designed (*Experiment 1* of Exercise 5), and explain why you have designed them the way you did;
3. report the results of the best algorithm (*Experiment 2* of Exercise 5);
4. outline ideas of how you would improve your work if you would have one more month.

In this video, you can use (but you don't have to use) tables, charts, animations, etc.

Do not submit the video file, **but only a link to the video**.

3 Marking

Marking of the implementation will be done according to the following criteria:

- correct overall implementation - 25%
- quality of the code (succinctness, object orientated, class structure) - 50%
- code comments - 25%

The following link contains general information that can be helpful:

- Feedback that we have given in the past:

  `https://cs.adelaide.edu.au/~markus/teaching/feedback.txt`

  (not everything is applicable here)

4 Procedure for handing in the assignment, and further requirements

There should be only one submission per group. We will inform you in time how you can make the submission.

The submission needs to include:

- all source files – if your code is insufficiently documented, **we will cap the code-related marks at 50%**
- all configuration files
- a README.txt file containing instructions to run the code, the names, student numbers, and email addresses of the group members – if this file is missing, **you will get 0 marks**
- for Exercise 5: the two plain text files showing the results (one file for the three algorithms, one file for the multiple runs of the "best" algorithm)
- the log-files containing the results generated by the code:
  - please generate short log-files that still show the overall optimisation (e.g., output the "best fitness in the current population" every 100 generations)
  - **if you do not provide log-files, you will receive 0 marks for Exercise 5**

Failure to meet all requirements of the 'General procedure for handing in the assignment' will lead to a reduction by twenty (20) marks.