Assignment 1: The Traveling Salesman Problem

Core Body of Knowledge classification: Abstraction (5), Design (5), Teamwork concepts & issues (3), Data (5), Programming (5), Systems development (3)

Due date: 11:59pm, 24 August 2019, Weight: 16.67 % of total assignment work

1 Overview

Assignments should be done in groups consisting of 5-6 students. Each student has to take major responsibility for one of the exercises and collaborate with the team members on the remaining exercises. Each exercise needs one student taking major responsibility. The group has to make sure that the workload is evenly distributed.

If you have problems finding a group use the forum to search for group partners or contact the lecturer. **It is compulsory that your group presents the preliminary results during the assignment presentation session.**

Each assignment has to be handed in and reasonable effort has to be made to complete each assignment in order to pass the course. For assignment 1 the minimum requirement is that the local search algorithms and different modules of evolutionary algorithms for the TSP have been implemented as described below and that evolutionary algorithms for the TSP have been designed and tested on the benchmarks given in TSPlib.

2 Assignment

The first assignment is to design a library and implement local search and evolutionary algorithms for the traveling salesman 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. Construct a system that can be extended by using different methods for each part of the design (for example different individual representations, operators, selection methods etc). 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 the account the recommendations given in the lecture.

All programming work should be done in JAVA.

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

Write a class *TSP-Problem* which represents the TSP problem. Your class should enable the construction of TSP problems from the files of the symmetric traveling salesman problem (EDGE_WEIGHT_TYPE : EUC_2D) of the TSPlib which is available at

`http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/index.html`

Exercise 2 *Local Search (20 + 10 marks)*

- Implement three local search algorithms based on jump, exchange and 2-opt neighbourhoods as described in the lecture.
- Test your local algorithms on the instances EIL51, EIL76, EIL101, ST70, KROA100, KROC100, KROD100, LIN105, PCB442, PR2392, USA13509 from TSPlib. Run each local search algorithm on each instance 30 times where in each run the initial permutation is chosen uniformly at random. Report for each algorithm on each instance the minimum and mean tour length obtained in a table. Summarize and compare the results obtained in 1-2 paragraphs.

Exercise 3 *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. A population in an evolutionary algorithm 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 4 *Variation operators (24 marks)*

- Implement the different mutation operators (insert, swap, inversion) 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 5 *Selection (15 marks)*

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

Exercise 6 *Evolutionary Algorithms and Benchmarking (26 marks)*

- Design three different evolutionary algorithms using crossover and mutation on the basis on the implementation of your different modules. Your algorithms should perform as best as possible. Explain your design choices.

- Test your algorithms with population sizes $20, 50, 100, 200$ on the instances EIL51, EIL76, EIL101, ST70, KROA100, KROC100, KROD100, LIN105, PCB442, PR2392, USA13509 from TSPlib. Report the outcomes after 2000, 5000, 10000, and 20000 generations.
- Run your best algorithm with a population size of 50 for 20000 generations on the TSPlib instances mentioned above. Run your algorithm on each instance 30 times. Report the average cost of the tour you obtained for each instance as well as the standard deviation.

Exercise 7 *Inver-over Evolutionary Algorithm (20 marks)*

- Read the paper "Inver-over Operator for the TSP" by Guo Tao, Zbigniew Michalewicz available at

  `http://www.cs.adelaide.edu.au/~zbyszek/Papers/p44.pdf`

- Implement the algorithm as described in the paper.
- Run the inver-over algorithm with a population size of 50 for 20000 generations on the TSPlib instances mentioned above. Run the algorithm on each instance 30 times. Report the average cost of the tour you obtained for each instance as well the standard deviation.

3 Marking

Marking will be done according to the following criteria:

- correct overall implementation 25%
- quality of the results 25%
- quality of the code (succinctness, object orientated, class structure) 10%
- modularity (extent different parts can be extended or exchanged, e.g. to add other operators or selection methods) 15%
- comments 15%
- configuration 10%

4 Procedure for handing in the assignment

Work should be handed in using Canvas. The submission should include:

- list showing for each exercise the student taking major responsibility.

- for each student 4-5 sentences summarizing their contribution to this assignment work.
- all source files
- all configuration files
- a text file called results.txt containing results generated by the code
- tables showing the results of your best local search algorithm, best evolutionary algorithm, and the inver-over algorithm on the considered TSP instances.
- a README.txt file containing instructions to run the code, the names, student numbers, and email addresses of the group members

In addition, there will be a discussion session where you have to explain your solutions.