CSE 8006 – Evolutionary Computation
Programming Project #1

DUE DATE: 28 / 10 / 2018 – 19:00

Demo Session: 1 / 11 / 2018 – 09:30

Travelling Salesman Problem (TSP) is a very popular problem, which aims to find a minimum-distance path between some number of cities that a salesman might follow on a business trip. The best path is the path that has the shortest total round-trip distance--from an arbitrary starting city, though all other cities, and back to the starting city. Unfortunately, finding the best path is an NP-complete problem. In this assignment, we target to develop a GA-based solution for TSP problem.

Details of GA-Based Solution

- **String Representation:** We will consider *path representation* where the order in a solution defines the order that the traveling salesman will visit the other cities.
- **Initialization:** 50% of the population is initialized randomly. For the remaining part, we consider nearest-neighbor strategy, a very simple and intuitive way for initialization. In this strategy, first a city *x* is selected randomly. Then, it selects the city that is closest to city *x* and that has not been selected yet; now the new city becomes the base and this step is repeated until all cities are selected. Note that it will prevent duplicates of selecting the first city "x".
- **Fitness Function**: It is defined as the total distance of travel represented by the order of cities. Note that we consider a closed TSP where it will begin and end to the same city.
- **Population Size**: Population size is set to 50, unless otherwise stated.
- **Crossover Operator**: You will implement two different crossover operators, which are the Order Crossover (OX) and Sequential Constructive Crossover (SCX). Literature survey is required. (ScienceDirect and/or IEEE Explore)
- **Mutation Operator:** There will be three types of mutation operators which are the insertion mutation (ISM), the inversion mutation (IVM) and the swap mutation (SM). The mutation probability is 10%, unless otherwise stated. It should be noted that once a mutation operator is selected, it will not be changed throughout the execution. The fourth alternative is to select a random mutation operator among the given three cases (ISM, IVM, SM), when the mutation operator is called in the program.
- **Parent Selection:** We will consider tournament selection with a tournament size of 5. Note that it will prevent duplicates for parent selection.
- **Survivor Selection.** You will implement a *steady-state GA*, where entire population is not changed at once, but a part of it is changed. One of the offspring will be written in place of the worst individual of the population and the other one is written in a random position in the population.
- **Termination Condition.** It will terminate after 10000 generations, unless otherwise stated.

Experimental Study

Note that as part of the project, a data file (i.e., a benchmark for the experiments) will be given. The name of the data file will be hardcoded; it will not be an argument. You are required to conduct the following four set of experiments and present the results. You are required to present the detailed analysis of results and corresponding figures/ charts (in Excel format) as well.

Experiment 1: Varying the Crossover and Mutation Types

For a population size of 50, run each case (a total of 8 cases) 100 times by using different seed values. Note that there will 8 cases to be considered; since 2 crossover and 4 mutation types are examined. Then, *present* the best result and average result (of the population) after 1000, 5000 and 10000 generations, for each case.

Experiment 2: Performance of GA with Respect to Number of Generation

In this experiment, you are required to consider only best and worst cases from the first experiment. For each of these two sets, run the algorithm once by measuring the performance per generation. You are required to monitor performance of the best solution and average solution of the population after each generation. Population size will be set to 50. Note that performance values will be stored for all generations up to 10000. You will present the results with figures which show performance with respect to generation number.

Experiment 3: Improving the Performance of Tours

In this case, we consider the best case from the first experiment and try to improve its performance by applying local search strategies.

Once in every *k generations/iterations* (k=20, unless otherwise stated), you are required to apply the **2-opt operator** on both the best individual from the population and randomly selected m individuals from the population (m=1, unless otherwise stated). Note that related procedure is called after kth iteration is completed. If the new tour generated by the 2-opt operator is shorter/better than the current one, the new one is accepted. Note that once in every "k" generations, 2-opt operator is applied on the selected individual n times (including both accepted and non-accepted cases). Here, n is equal to 3, unless otherwise stated.

2-opt operator = It removes two randomly selected edges from the tour, and then it reconnects the two paths created. There is only one way to reconnect the two paths so that the result is also a valid tour.

Consider the following values for k,m,n parameters, which causes a total of 9 test cases: k={10,20,50}, m={1,2,5} and n={3,5,10}. Run each case 100 times and present the best result, the worst result and average result of the population after 10000 generations. For each case, present the amount of improvement over the baseline case which does not utilize 2-opt operation.

What to Submit:
Note that all programs should be written in C programming language. You are required to submit i) *program listing (source codes)*, ii) *a research paper (> 5 pages) which summarizes the problem, implementation details and results of experiments.* A file for benchmark data and a document which has related information will be emailed to you.