INF3490/INF4490 Mandatory Assignment 1: Travelling Salesman Problem

Eivind Samuelsen[*]

Rules

Before you begin the exercise, review the rules at this website:

`http://www.uio.no/english/studies/admin/compulsory-activities/mn-ifi-mandatory.html`

(This is an individual assignment. You are not allowed to deliver together or copy/share source-code/answers with others.)

Delivering

Deadline: Monday, September 25th, 2017 (25.09.2017) 23:59:00 All deliveries must be made through **Devilry**.

What to deliver?

Deliver one single zipped folder (.zip, .tgz or .tar.gz) which includes:

- PDF report containing:
  - Your name and username (!)
  - Instructions on how to run your program.
  - Answers to all questions from assignment.
  - *Brief* explanation of what you've done.
- Source code
- The european cities file so the program will run right away.
- Any files needed for the group teacher to easily run your program on IFI linux machines.

Important: if you weren't able to finish the assignment, use the PDF report to elaborate on what you've tried and what problems you encountered. Students who have made an effort and attempted all parts of the assignment will get a second chance even if they fail initially. This exercise will be graded PASS/FAIL.

Introduction

In this exercise, you will attempt to solve an instance of the traveling salesman problem (TSP) using different methods. The goal is to become familiar with evolutionary algorithms and to appreciate their effectiveness on a difficult search problem. You may use whichever programming language you like, but we strongly suggest that you try to use Python, since you will be required to write the second assignment in Python. You must write your program from scratch (but you may use non-EA-related libraries).

[*]See **README** for complete list of authors/contributors.

	Barcelona	Belgrade	Berlin	Brussels	Bucharest	Budapest
Barcelona	0	1528.13	1497.61	1062.89	1968.42	1498.79
Belgrade	1528.13	0	999.25	1372.59	447.34	316.41
Berlin	1497.61	999.25	0	651.62	1293.40	689.06
Brussels	1062.89	1372.59	651.62	0	1769.69	1131.52
Bucharest	1968.42	447.34	1293.40	1769.69	0	639.77
Budapest	1498.79	316.41	689.06	1131.52	639.77	0

Table 1: First 6 cities from csv file.

Problem

The traveling salesman, wishing to disturb the residents of the major cities in some region of the world in the shortest time possible, is faced with the problem of finding the shortest tour among the cities. A tour is a path that starts in one city, visits all of the other cities, and then returns to the starting point. The relevant pieces of information, then, are the cities and the distances between them. In this instance of the TSP, a number of European cities are to be visited. Their relative distances are given in the data file found at `http://www.uio.no/studier/emner/matnat/ifi/INF3490/h17/assignment-1/european_cities.csv`.
(You will use permutations to represent tours in your programs. If you use Python, the itertools module provides a permutations function that returns successive permutations, this is useful for exhaustive search)

Exhaustive Search

First, try to solve the problem by inspecting every possible tour. Start by writing a program to find the shortest tour among a subset of the cities (say, 6 of them). Measure the amount of time your program takes. Incrementally add more cities and observe how the time increases.

What is the shortest tour (i.e., the actual sequence of cities, and its length) among the first 10 cities (that is, the cities starting with B,C,D,H and I)? How long did your program take to find it? How long would you expect it to take with all 24 cities?

Hill Climbing

Then, write a simple hill climber to solve the TSP. How well does the hill climber perform, compared to the result from the exhaustive search for the first 10 cities? Since you are dealing with a stochastic algorithm, you should run the algorithm several times to measure its performance. Report the length of the tour of the best, worst and mean of 20 runs (with random starting tours), as well as the **standard deviation** of the runs, both with the 10 first cities, and with all 24 cities.

Genetic Algorithm

Next, write a genetic algorithm (GA) to solve the problem. Choose mutation and crossover operators that are appropriate for the problem (see chapter 3 of the Eiben and Smith textbook). Choose three different values for the population size. Define and tune other parameters yourself and make assumptions as necessary (and report them, of course).

For all three variants: As with the hill climber, report best, worst, average and deviation of tour length out of 20 runs of the algorithm (of the best individual of last generation). Also, find and plot the average fitness of the best fit individual in each generation (average across runs), and include a figure with all three curves in the same plot in the report. Conclude which is best in terms of tour length and number of generations of evolution time.

Among the first 10 cities, did your GA find the shortest tour (as found by the exhaustive search)? Did it come close? For both 10 and 24 cities: How did the running time of your GA compare to that of the exhaustive search? How many tours were inspected by your GA as compared to by the exhaustive search?

Hybrid Algorithm (INF4490 only)

Implement a hybrid algorithms to solve the TSP: Couple your GA and hill climber by running the hill climber a number of iterations on each individual in the population as part of the evaluation. Test both Lamarckian and Baldwinian learning models and report the results of both variants in the same way as with the pure GA (min, max, mean and std.dev. of the end result and an averaged generational plot). How does the results compare to that of the pure GA, considering the number of evaluations done?

Contact and Github

Corrections of grammar, language, notation or suggestions for improving this material are appreciated. E-mail me at **tjbersta@uio.no** or use **GitHub** to submit an issue or create a pull request. The **GitHub repository** contains all source code for assignments, exercises, solutions, examples etc. As many people have been involved with writing and updating the course material, they are not all listed as authors here. For a more complete list of authors and contributors see the **README**.