IN3050/IN4050 Mandatory Assignment 1: Traveling Salesman Problem

samesra@uio.no - Sameer Ahmed

Rules

Before you begin the exercise, review the rules of the exercise: https://www.uio.no/studier/emner/matnat/ifi/IN3050/v22/obligatorisk-oppgave-1/index.html#rules (This is an individual assignment. You are not allowed to deliver together or cooperate across collaboration with others.)

Pay special attention to the rules for how to deliver your solution. Deliver your solution in a zip file. Include in the zip file: (1) a short note specifying how to **unzip all the right files and run your main programs/scripts** and (2) a PDF or HTML version of your report. Please use the Fronter assignment mandatory delivery and attached checking of the University of Oslo? (https://www.uio.no/english/studies/examinations/digital-exams/fronter/index.html) By submitting this assignment, you confirm that you understand the rules for the assignment in IN3050/IN4050.

Deadline

Deadline: Friday, February 25 2022, 23:59

Recommendation should be IN3050/IN4050. You may weather a Gravity before the deadline, but include of this in the text delivery, as only the text delivery will be read. You are recommended to upload preliminary versions hours or days before the final deadline.

What to deliver?

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

- PDF report containing:
  - Your name and username (if applicable)
  - Instructions on how to unzip and run your scripts with example runs.
  - Answers to all questions that assignment.
  - Your implementation of your best idea.
  - Short description of your best idea implementing your Jupyter Notebook in PDF, if you have answered all questions in your notebook
- Source code:
  - Your implementation of your best idea, preferably as a runnable Nav.ipynb
  - The necessary utilities for the program will run right away.
  - Instructions for unzipping the zipping folder and how to run your program on Ifi-Linux machines.

Important:

- Include executable source all code used for doing the results described in the notes. Simply implementing the code, but never running it will not give many points.
- It is fine to use tools that can speed up your work, for example using ChatGPT to generate source all functions and the others also including the code that you used to produce it.
- Clearly indicate which parts of the code you have written yourself, which parts are based on code you have found on the internet, and which parts are based on others your attempted ideas of the assignment will get a second chance even if they had failed. 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 stochastic algorithms and to appreciate their effectiveness in a difficult search problem. Your task is to find a tour of Norway that is shorter than the tour in the example provided. You must write your program from scratch but you may use any of the related libraries.

Helper code for visualizing solutions

Here follows some Helper code that you can use to visualize the plots you generate. These visualizations can **help you check if you are making sensible tours or not**. The visualizations are based on the matplotlib library, and the functions rely on having a list of cities with latitudes and longitudes.

```
import matplotlib.pyplot as plt
import numpy as np
import math
import random
import time
from IPYTHON import *
%matplotlib inline

# Plot all the coordinates
# cities: {"City": (latitude, longitude)}
cities = {"Oslo": (59.9127, 10.7461), "Bergen": (60.39, 5.32), "Trondheim": (63.43, 10.39),
          "Stavanger": (58.97, 5.73), "Tromsø": (69.65, 18.96), "Kristiansand": (58.14, 7.99),
          "Tynset": (62.27, 10.78), "Hønefoss": (60.17, 10.25), "Hamar": (60.79, 11.07),
          "Ålesund": (62.47, 6.15), "Molde": (62.74, 7.15), "Lillehammer": (61.11, 10.46),
          "Røros": (62.58, 11.38), "Narvik": (68.44, 17.42), "Kirkenes": (69.73, 30.04),
          "Leirfjord": (66.02, 13.04), "Leknes": (68.15, 13.61), "Svolvær": (68.24, 14.57),
          "Vardø": (70.37, 31.03), "Mosjøen": (65.84, 13.19), "Namsos": (64.47, 11.49),
          "Florø": (61.60, 4.85), "Arendal": (58.46, 8.77), "Kristiansund": (63.11, 7.73),
          "Steinkjer": (64.01, 11.50), "Tromsø": (69.65, 18.96), "Hammerfest": (70.66, 23.68),
          "Alta": (69.97, 23.27), "Bodø": (67.28, 14.38), "Mosjøen": (65.84, 13.19),
          "Sandnessjøen": (66.02, 12.63), "Mo i Rana": (66.31, 14.14), "Honningsvåg": (71.00, 25.97),
          "Vardø": (70.37, 31.03), "Kirkenes": (69.73, 30.04), "Hammerfest": (70.66, 23.68),
          "Tromsø": (69.65, 18.96), "Tromsø": (69.65, 18.96
```

```
plt.plot(avg_fitness_50, label="Population 50")
plt.plot(avg_fitness_100, label="Population 100")
plt.plot(avg_fitness_200, label="Population 200")
plt.xlabel("Generation")
plt.ylabel("Average Best Fitness")
plt.title("Average Best Fitness over Generations for 24 Cities")
plt.legend()
plt.grid(True)
plt.show()
```

```
average time was 0.06378679275512696
Best for 10 cities: 7486.309999999995
Worst for 10 cities: 7680.299999999999
Mean for 10 cities: 7496.648999999999
Standard Deviation for 10 cities: 42.24488950157155 6
```

```
Population 50:
average time was 0.09550776481628417
Best for 24 cities: 13224.519999999999
Worst for 24 cities: 17471.04
Mean for 24 cities: 14882.6785
Standard Deviation for 24 cities: 1024.9633439068687
```

```
Population 100:
average time was 0.17613929510116577
Best for 24 cities: 12605.08
Worst for 24 cities: 16191.089999999997
Mean for 24 cities: 13990.259500000004
Standard Deviation for 24 cities: 803.9671114695856
```

```
Population 200:
average time was 0.18335341215133666
Best for 24 cities: 12874.35
Worst for 24 cities: 15996.32
Mean for 24 cities: 14116.613000000001
Standard Deviation for 24 cities: 794.7414054716165
```

Choose three different values for the population size. Define and tune other parameters yourself and make assumptions as necessary (and report them, of course):

Answer:

We can see for the different population sizes that there is a big difference from the 50 in population to the 100 and 200, that are closer in distances. Surprisingly the 100 in population size got the shortest route. We can see form the curve that the 100 and 200 was close until generation 65, where the 100 in population size got a big advantage. From there it got closer until the end, but the 100 in population size got the shortest route.

I also tried different mutation rates, but I found that around 0.3 was the best for my genetic algorithm.

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?

In [ ]:
```
# Answer

# Yeah, my GA found the shortest tour with a population size of 100 in 50 generations.

# My GA used an average of 0.063 seconds for 10 cities, compared to the exhaustive search that used 4.86 seconds
#For 24 cities my GA used an average of around 0.18 with population sizes of 100 and 200.

# For 10 cities the GA with a population size of 100 and 50 generations, it would inspect around 3350.
# 100 for initial population, 50 children and around 15 mutations per generation.
# The exhaustive search inspects 10!
```

Hybrid Algorithm (IN4050 only)

Lamarckian

Lamarck, 1809: Traits acquired in parents' lifetimes can be inherited by offspring. In general the algorithms are referred to as Lamarckian if the result of the local search stage replaces the individual in the population.

Baldwinian

Baldwin effect suggests a mechanism whereby evolutionary progress can be guided towards favourable adaptation without the changes in individual's fitness arising from learning or development being reflected in changed genetic characteristics. In general the algorithms are referred to as Baldwinian if the original member is kept, but has as its fitness the value belonging to the outcome of the local search process.

(See chapter 10 and 10.2.1 from Eiben and Smith textbook for more details. It will also be lectured in Lecure 4)

Task

Implement a hybrid algorithm 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 standard deviation of the end result and an averaged generational plot). How do the results compare to that of the pure GA, considering the number of evaluations done?

In [ ]:
```
# Implement algorithm here
```