Assignment #2

Assignment Overview

In this assignment you will implement an evolutionary approach to solving the classical Traveling Salesman Problem (TSP) using genetic algorithms (GA) in Python.

Background

This assignment will give you a chance to build a working solution for the TSP problems and demonstrate your knowledge of genetic algorithms. See chapters 4 and 5 of the textbook for additional details.

You can learn more about the TSP by following the links below:

- https://en.wikipedia.org/wiki/Travelling_salesman_problem
- http://www.math.uwaterloo.ca/tsp/
- https://www.quantamagazine.org/computer-scientists-break-traveling-salesperson-record-20201008/

If you're feeling adventurous, there is an entire book[1] about it, whose first chapter is available on Canvas.

Solving the TSP using genetic algorithms: recommendations

This assignment will focus on implementing a Python solution using GAs to solve the TSP. To help you navigate the design process and find good "sources of inspiration" out there, here are a few **recommendations**:

1. Start by revising your understanding of GAs (in general) and planning how to attempt to solve the TSP in a GA manner.
   - For example: What are the main steps in a classical GA and how does the TSP differ from the various flavors of the knapsack algorithm discussed in the textbook?

2. Consider looking at "brute force" solutions to the problem, such as this: https://www.geeksforgeeks.org/traveling-salesman-problem-tsp-implementation/

3. Break the problem down into parts and **provide explicit answers to these questions in your report**:
   - How were the **cities and distances represented (as a data structure)?**
   - How did you encode **the solution space**?
   - How did you handle the **creation of the initial population**?
   - How did you compute the **fitness score**?
   - Which **parent selection strategy** did you use? Why?
   - Which **crossover strategy**(ies) did you try? Which one worked out best?
   - Which **mutation strategy**(ies) did you try? Which one worked out best?
   - Which **strategy** did you use **for populating the next generation**? Why?
   - Which **stopping condition** did you use? Why?
   - What **other parameters, design choices, initialization and configuration steps** are relevant to your design and implementation?
   - Which (simple) **experiments** have you run to observe the impact of different design decisions and parameter values? Post their results and your comments.

---

[1] Applegate, David L., et al. *The traveling salesman problem*. Princeton University Press, 2011. https://www.degruyter.com/document/doi/10.1515/9781400841103/html

4. Use *good code* as starting point (and make sure to reference them accordingly in your report!)
   - Examples:
     - Code from the (GAIA) textbook (chapters 4 and 5)
     - OOP solution (in Java) by Lee Jacobson: https://www.theprojectspot.com/tutorial-post/applying-a-genetic-algorithm-to-the-travelling-salesman-problem/5
     - OOP solution (in Python) by Eric Stoltz: https://towardsdatascience.com/evolution-of-a-salesman-a-complete-genetic-algorithm-tutorial-for-python-6fe5d2b3ca35
     - https://thecodingtrain.com/CodingChallenges/035.1-tsp (video series + many links)

Project Specification

This is a group assignment.
Students are encouraged (but not required) to work in groups of max 3 students.
Ideally, the group should be organized around three main tasks / duties:

- Design of the solution ("architect" role)
- Coding of the solution ("developer" role)
- Documentation of the solution ("reporter" role)

You are required to indicate in your report "who did what" and document the entire process, from sketching the original plans and dividing up the tasks all the way to polishing the interface, testing the solution, and preparing the report.

Deliverables

You must submit (via Canvas):

- **One** file **a1_FAUusername.py** (where "**FAUusername**" is the username of one of the team members); in my case the file would be called **a1_omarques.py**) that allows me to call the code for either solution.
  - This is your source code solution; be sure to include your names, date, assignment number and comments describing your code.
  - Only one submission per team is enough, if the names of the team members appear in the comments and report.

- (Optional) A **README.md file** with installation instructions, dependencies, etc.

- A **detailed report** (PDF, markdown, and/or HTML) with detailed "project notes" (describing what my TA and I cannot see by looking at your source code and/or running your program), screenshots, references, etc.
  - Examples: design decisions, documented limitations, future improvements, etc.
  - Your report must include explicitly your answers to the questions in the previous page.

- **Screenshots of representative results** produced by your code.
  - Make sure to show input and output for **at least 5 runs with different parameters.**

- **You are strongly encouraged to use Jupyter notebooks and Google Colab for this assignment!**
  - **If you build a polished notebook (and make it available via Colab) with your solution it will take care of all the aspects above (source code, README, plots, report, multiple runs) and you will earn up to extra 10 points!**

Notes and Hints:

- Try to handle special cases and prevent runtime errors to the best of your knowledge.
- **Don't overdo it!** You might want to try some of the bonus points options below but try not to risk breaking a good solution by adding bells and whistles to it.
- **Data for testing:** You should create a list of N cities (where the default for N = 25) within an (x, y) plane of 200-by-200 (think of them as km). Cities should be placed randomly within this space. Please ensure reproducibility by setting the seed of your random number generator to a fixed value of your choice. See example below for a code snippet:

```
cityList = []

for i in range(0,25):
    cityList.append(City(x=int(random.random() * 200), y=int(random.random() * 200)))
```

Bonus opportunities:

If you successfully implement one or more of the bonus options below **without breaking the baseline solution** (*) you will earn extra points (max 10% each):

1. **Web-based solution**
   - If you implement an **elegant** web-based solution (e.g., using Flask[2]) for the problem, you will earn up to extra 10 points per problem.

2. **Jupyter notebook / Google Colab**
   - If you build a polished notebook (and make it available via Colab) with your solution (and report?), you will earn up to extra 10 points.

(*)
A common mistake students make is to attempt bonus items before producing a "perfect 100" baseline solution.

Make no mistake: if your baseline solution doesn't meet **all** grading criteria (see rubric on Canvas), you are not eligible for bonus points. Period.

[2] https://flask.palletsprojects.com/en/2.0.x/