CS451/551 – Introduction to Artificial Intelligence
Assignment-2

Solving Travelling Salesman Problem with Genetic Algorithm

created by Furkan Kınlı, for help → e-mail: furkan.kinli@{ozu, ozyegin}.edu.tr
Language: Python or Java (choose wisely ☺)

1. Definition

In this assignment, you will implement Genetic Algorithm (GA) to solve one of the most well-known Computer Science problem, which is Traveling Salesman Problem (TSP). We will provide some baseline code for TSP Environment (on LMS), and you need to incrementally implement GA to solve TSP. Mainly, we expect you to implement major components of GA, which are crossing-over, mutation, evolving, tournament.

In code baseline, you will have 6 different script files for 2 languages as follows:

- main.{py, java} → Runner script, you are NOT allowed to change this file.
- City.{py, java} → Class file for City implementation, you should check over this file before starting to implement the algorithm to understand what a city can do. Preferred NOT to change, but you MAY.
- CityManager.{py, java} → Class file for managing City implementation, contains some utilities for City & Cities, you should check over this file before starting to implement the algorithm to understand how can manage a city. Preferred NOT to change, but you MAY.
- Route.{py, java} → Class file for Route implementation, you should check over this file before starting to implement the algorithm to understand what a Route can do. Preferred NOT to change, but you MAY.
- RouteManager.{py, java} → Class file for managing Route implementation, contains some utilities for Route & Routes, you should check over this file before starting to implement the algorithm to understand how can manage a route. Preferred NOT to change, but you MAY.
- GeneticAlgorithmSolver.{py, java} → Class file for solving GA, you need to implement particular methods to design this algorithm.
    Methods → Return value
    - evolve(list of routes) → list of routes
    - crossover(route1, route2) → route
    - mutate(route) → void
    - tournament(list of routes) → route

2. Traveling Salesman Problem

The travelling salesman problem (TSP) asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?" It is an *NP-hard* problem in combinatorial optimization, important in operations research and theoretical Computer Science.

The problem was first formulated in 1930s and is one of the most intensively studied problems in optimization. It is used as a benchmark for many optimization methods. Even though the problem is computationally difficult, many heuristics and exact algorithms are known, so that some instances with tens of thousands of cities can be solved completely and even problems with millions of cities can be approximated within a small fraction of 1%. TSP is a touchstone for many general heuristics devised for combinatorial optimization such as genetic algorithms, simulated annealing, tabu search, ant colony optimization, river formation dynamics (see swarm intelligence) and the cross-entropy method.

For more information:

- Wiki
- *Travelling Salesman*, by director Timothy Lanzone, is the story of four mathematicians hired by the U.S. government to solve the most elusive problem in computer-science history: P vs. NP.[64]
- Traveling Salesman Problem at the Wayback Machine (archived 2013-12-17) at University of Waterloo
- TSP visualization tool

3. Genetic Algorithm

In Computer Science and operations research, a genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover and selection. John Holland introduced genetic algorithms in 1960 based on the concept of Darwin's theory of evolution; his student David E. Goldberg further extended GA in 1989.

For more information:

- Wiki
- Genetic Algorithms - Computer programs that "evolve" in ways that resemble natural selection can solve complex problems even their creators do not fully understand An excellent introduction to GA by John Holland and with an application to the Prisoner's Dilemma
- An online interactive Genetic Algorithm tutorial for a reader to practise or learn how a GA works: Learn step by step or watch global convergence in batch, change the population size, crossover rates/bounds, mutation rates/bounds and selection mechanisms, and add constraints.
- Genetic Algorithms in Python Tutorial with the intuition behind GAs and Python implementation.

I strongly recommend & ask you to read about the algorithm and the problem before starting to implement your assignment. Provided class files should be investigated before mailing to TA. TA will NOT help coding!

4. Implementation Details

As we provided a code baseline, you have to be strict on this baseline. You may propose some other implementation for this assignment, but this is your responsible to implement the environment and algorithm correctly, and TA cannot help in detail (tries, but ☺). City and Route implementations are given to you to deeply understand what they actually do, and suggest you to not change these files. Also, you will have a runner script which is named as "main.py", and this file is basically responsible for running your script.

You need to implement GA algorithm on different components in corresponding class file (*e.g.* GeneticAlgorithmSolver.py or .java), and report your results accordingly. Please first try to understand the main structure of the environment, and then start to implement the algorithms. Please try to NOT change any code block unless it has the comment line as

"YOUR CODE HERE"

If you are not comfortable to write code in Python, you may follow this tutorial to improve your Python programming skills. Even if you do not know anything about Python, you only need 2-3 days to learn Python from scratch to be able to complete this assignment. If you have any question about the implementation, please do not hesitate to send an e-mail to the assistant.

This algorithm and this problem are well-studied in the literature and online resources, and there are many different coding examples online. We have almost all of them, and please do NOT try to use them directly. In any circumstances of copying online resources directly, you will get 0 point.

5. Criterion

Runtime: Please do NOT include any third-party library to your project (except NumPy), because you do not need any extra library. Including different third-party library may cause a problem for running your code in different local machines (*e.g.* dependencies etc.).

For any dependency (third-party library) on runtime that the assistant can solve → -15 points penalty.

for any dependency on runtime that the assistant cannot solve → -30 points penalty.

Compilation: Please make sure that your code is compiled properly.

Not compiling code (*e.g.* any error etc.) → --60 points penalty.

Report: You need to write a report to explain your design in detail. 1 paragraph code explanation is not a report, and such reports will be ignored. Please follow the rules of technical report writing (*e.g.* Introduction, Algorithms, Implementation Details, Results, Conclusion etc.).

Submission: All project files (*.PY or *.JAVA) and report file (.PDF) should be zipped (.ZIP) together, and the filename of ZIP file should be in the format of "NAME_SURNAME_ID_hw2.zip". Please follow this structure on your submission.

Not following → -10 points penalty.

Due date: 10 April 2020 – 23.55 via LMS.
Late submission: Allowed for 3 extra days, with -10 for 1st day, -15 for 2nd day, -20 for 3rd day.

Group effort: You may discuss the algorithms and so forth with your friends, but this is and individual work, so you have to submit your original work. **In any circumstances of plagiarism, first, you will fail the course, and the necessary actions will be taken immediately.**

Grading criteria:

- Evolve: 25 points
- Crossing-over: 20 points
- Mutation: 20 points
- Tournament (Selection): 20 points
- Report: 15 points