CSE 140: Assignment #4

Summer 21

DUE Sunday, 18 JULY 23:59:59

MAX POINTS: 100. Released: — July 14, 2021

General Instructions

This assignment is split into three parts. The starter code for all three parts is given to you in a single file named `asg4.py`. You need to implement the required functions in it. The instructions for each part of the assignment is described below. You are also required to turn in a report named `report.pdf`. This report should contain your `NAME`, `CruzID` (email), and `StudentID` with a few paragraphs of content. The content of the report is described later in this pdf. You need to submit these two files `asg4.py` and `report.pdf` to Canvas.

Below are the instructions for the different parts of the assignment.

1 Part 1: Travelling Salesperson Problem

In this part of the assignment you are required to implement the Travelling Salesperson Problem using the three local search optimization algorithms mentioned below.

- Hill-Climbing
- Random Restart Hill-Climbing
- Stochastic Hill-Climbing

1.1 Objective

In the first part you will be working with the class `TSP_GRAPH`. Your objective is to implement the Travelling Salesperson Problem using the local search optimization algorithms mentioned above. The inputs for the Travelling Salesperson Problem is a graph which is represented in the files named `tsp_graph_N.txt` where `N` is a number which differentiates the different input files. You are given multiple input files to test your code.

1.2 CODE

This section briefly describes the starter code that is given to you for this part of the assignment. As mentioned above you will be working with the class `TSP_GRAPH`. Comments describing the functions and class are provided in the code. This class has several functions. They are:

1.2.1 `__init__`

This is the constructor for the `TSP_GRAPH` class. It initializes the graph and the number of vertices to 0. There are two variables, `V` denoting the number of vertices and `graph` used to represent the graph. The graph is represented as a list of lists. The first inner list represents the first vertex and all it's edges to all other vertices, the second inner list represents the second vertex and all it's edges to all other nodes and so on. The value `0` is used to represent the fact that there are no edges from a vertex to another. The graph can be initialized in multiple ways as described in the comments above the function.

1.2.2 `get_graph`

This function is used to read the graph from a file and store it in the `self.graph` variable as well as storing the number of vertices in the graph in the variable `self.V`. This function takes a number `N` as input, using which it decides which file to get the graph from.

1.2.3 `hill_climbing`

This function is where you need to implement the Travelling Salesperson Problem using hill-climbing. You are required to start at the first node in the list and make your way back to the same node (start and end nodes) to find a solution. Neither a solution nor a globally optimal solution is guaranteed. You are allowed to add parameters and helper functions as needed to achieve this functionality.

1.2.4 `random_hill_climbing`

This is the function where you need to implement random restart hill climbing. You are required to start at the first node in the list and make your way back to the same node (start and end nodes). Neither a solution nor a globally optimal solution is guaranteed. You are required to use python's `random` module's `randint` or `random` functions to get random values. You are allowed to add parameters and helper functions as needed to achieve this functionality.

1.2.5 `stoch_hill_climbing`

This is the function where you implement the Travelling Salesperson Problem using stochastic hill climbing. You are required to start at the first node in the list and make your way back to the same node (start and end nodes). Neither a solution nor a globally optimal solution is guaranteed. You are required to use python's `random` module's `randint` or `random` functions to get random values. You are allowed to add parameters and helper functions as needed to achieve this functionality.

1.2.6 Evaluation

You are allowed to use any of the input files named `tsp_graph_N.txt` to evaluate your code.

2 Part 2: Job-Shop Problem

In this part of the assignment, you are required to implement the Job-Shop problem using the optimization algorithms mentioned below.

- Simulated Annealing
- Genetic Algorithms

2.1 Objective

In this part of the assignment you will be working with the class `JOB_GRAPH`. Your objective is to implement the Job-Shop problem using Simulated Annealing and Genetic Algorithm. The inputs for the Job-Shop problem are represented in files named `job_graph_N.txt` where `N` is a number differentiating the different input files. You are given multiple input files to test your code.

2.2 CODE

This section briefly describes the starter code that is given to you for this part of the assignment. As mentioned above you will be working with the class `JOB_GRAPH`. Comments describing the functions and class are provided in the code. This class has several functions. They are:

2.2.1 `__init__`

This is the constructor for the class `JOB_GRAPH`. It initializes three variables to zero/empty. These variables are `self.NJ` which represents the total number of jobs, `self.NM` which represents the total number of machines, and `self.jobs` which is used to store the different jobs. The `self.jobs` is a list of lists of sets. The inner lists represent a particular job, for example the first inner list represents `job1` and the second inner list represents `job2` and so on. The internal structure of the inner lists is a tuple/set. The two values represented are the `MachineID` and the `ProcessingTime` for each step of the job. All these are initialized to zero/empty.

2.2.2 `get_jobs`

This function reads a file using a parameter that is passed to it. The parameter passed to it is `N`, using which it determines which file to read. After reading the file, it sets the `self.NJ`, `self.NM`, and `self.jobs` variables mentioned in the `__init__` section. The file format is as follows. Each file's first line has the total number of jobs and total number of machines and every line after that corresponds to a particular job. Within each job line, the machineID and the Processing time for that step of the job is listed.

2.2.3 `sim_anneal`

This is where you need to write the Simulated Annealing version of the Job-Shop problem. The parameters are as follows. Use a starting temperature of `500`. Change the temperature in `0.5` increments. Use stagnation to terminate the code after `10,000` steps of no gain achieved. Use appropriate values for other parameters if necessary. You are allowed to add parameters and helper functions to achieve this functionality.

2.2.4 `genetic`

This is where you need to write the Genetic Algorithm based version of the Job-Shop problem. You need to use random selection to select the parents. You need to use a random mutation probability of `0.75`. Terminate the code after `10,000` generations. You are required to use python's `random` module's `randint` or `random` functions to get random values. You are allowed to add parameters and helper functions as needed to achieve this functionality.

2.2.5 Evaluation

You are allowed to use any of the input files `job_graph_N.txt` to evaluate your code.

3 Part 3: Graph Coloring Problem

This is the final part of the assignment. You will be working with the class `COLOR_GRAPH`.

3.1 Objective

You are to implement the Graph Coloring problem. As mentioned above, you'll be working with the class `COLOR_GRAPH`. The input for this part is in a single file named `color_graph.txt`. For this part of the assignment, we will just use the generic colors `color1`, `color2`, `color3` and so on to color the graph. This is so that different students don't use different colors, causing confusion when grading. You the results for this part need to be generated in the form of a list called `color_list`. This list contains the color assigned to each vertex of the `COLOR_GRAPH` in order.

Once you generate the `color_list` mentioned above, use the color assigned to call the functions implemented in **Part 1** and **Part 2** in the manner described below, measuring performance using python's `time` module.

- Iterate through the list of assigned colors and for each color in position N, based on the color assigned for the node, call the functions as described in the below **CASEs**.

- **CASE 1:** If `color1` was assigned, call the different implementations of the Travelling Salesperson Problem using a graph represented in `tsp_graph_N.txt`.
- **CASE 2:** If `color2` was assigned, call the different implementations of the Job-Shop problem using a graph represented in `job_graph_N.txt`.
- **CASE 3:** If `color3` was assigned, call the different implementations of the Travelling Salesperson Problem using graph represented in `tsp_graph_N.txt` and measure performance and if the solution was reached and if it was the globally optimal one. Compare your performance results in a few paragraphs in the `report.pdf`. Also mention your implementation choices when implementing the aforementioned algorithms for each implementation.

3.2 CODE

You will be working with the class `GRAPH_COLOR`. This class has several functions as described below.

3.2.1 `__init__`

This is the constructor for the `GRAPH_COLOR` class. It initializes the graph and number of vertices to 1. The graph is stored as a list of lists where the first inner list represents the first vertex, the second inner list represents the second vertex and so on. The values of the inner list can be `1` representing an edge between the two vertices or `0` representing no edge between two vertices.

3.2.2 get_graph

This function reads a file passed to it as a parameter and stores the total number of vertices in that graph and the graph.

3.2.3 do_color

This is the function where you need to implement the graph coloring algorithm. You can add parameters and helper functions to achieve this.

You can implement the functionality of calling different functions after generating the `color_list` from this function or return the `color_list` to main and call the functions from there.

4 SUMMARY

To recap, you are required to submit two files.

1. All your code in `asg4.py` written in `python3`, with comments about your implementation choices.
2. A pdf file named `report.pdf` containing your **NAME**, **CruzID**(email), **StudentID**, and the contents of your report; a few paragraphs going over your implementation choices and comparing the performance of the different **CASES** mentioned in this document.

Here are some additional things you should keep in mind.

- You need to implement the Graph Coloring Problem. The input for this is in the file `tsp_graph_N.txt`.
- You need to implement three versions of the Travelling Salesperson Problem using Hill-Climbing, Random Restart Hill-Climbing, and Stochastic Hill-Climbing.
- You need to implement two versions of the Job-Shop Problem using Simulated Annealing and Genetic Algorithm.
- The inputs for the Travelling Salesperson Problem are in multiple files named `tsp_graph_N.txt` where `N` is a number.

- The inputs for the Job-Shop Problem are in multiple files named `jsp_graph_N.txt` where `N` is a number.
- When writing the report, please mention your implementation choices, where you timed the different function calls and your experience when running the code, ie. how the different functions performed.