Homework #8
Search Tool (Version 4)

AI Programming

Overview

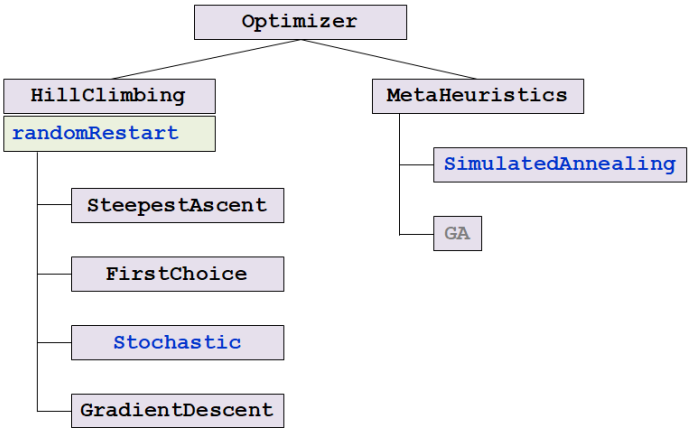
Your assignment this time is to add three more search algorithms: stochastic hill climbing, random-restart, and simulated annealing. This requires a more general class hierarchy to cover all the variety of algorithms. Also, due to this variety and related parameter settings, the user interface is better revised so that the setup information are read from a file provided by the user. A typical outcome of numerical optimization after revision would look like the following.

```
Enter the file name of experimental setting: exp.txt

Objective function:
20 + math.e - 20 * math.exp(-(1/5) * math.sqrt((1/5) * (x1 ** 2 + x2
** 2 + x3 ** 2 + x4 ** 2 + x5 ** 2))) - math.exp((1/5) * (math.cos(2
* math.pi * x1) + math.cos(2 * math.pi * x2) +math.cos(2 * math.pi
* x3) +math.cos(2 * math.pi * x4) +math.cos(2 * math.pi * x5)))

Search space:
 x1: (-30.0, 30.0)
 x2: (-30.0, 30.0)
 x3: (-30.0, 30.0)
 x4: (-30.0, 30.0)
 x5: (-30.0, 30.0)

Number of experiments: 5

Search Algorithm: First-Choice Hill Climbing

Number of random restarts: 10

Mutation step size: 0.01
Max evaluations with no improvement: 100 iterations

Average objective value: 17.441
Average number of evaluations: 16,232

Best solution found:
(2.992, 6.987, -1.0, 5.991, -10.985)
Best value: 14.623

Total number of evaluations: 81,162
```

Expanding to 'Optimizer' Class

Since the simulated annealing algorithm cannot belong to 'HillClimbing', you need to expand the class hierarchy. One possible way is to create a new superclass 'Optimizer' and let it have two subclasses 'HillClimbing' and 'MetaHeuristics'. Then, 'HillClimbing' will have four child classes: 'SteepestAscent', 'FirstChoice', 'Stochastic', and 'GradientDescent'. 'MetaHeuristics' will have two: 'SimulatedAnnealing' and 'GA' (In this lecture, we do not implement). The random-restart algorithm can be implemented as a method of 'Hill-

Climbing', which can be inherited to all the child hill climbers.



Notice that 'randomRestart' is a wrapper around any hill-climbing algorithm, which runs the algorithm repeatedly for a given number of times. Random restart can be easily made if each hill climber has not only a 'run' method for its own algorithm but also a 'randomRestart' method as a wrapper that keeps calling the 'run' method. However, instead of having the same 'randomRestart' method in duplicate in all the individual hill-climber subclasses, a better solution would be to make it a method of the parent 'HillClimbing' class and have it inherited to all the subclasses.

New User Interface

See the attached file 'exp.txt', which is an example file containing information about the experimental setup. The lines starting with the '#' symbol are like comments in Python programs. These lines just explains to the user what information he or she should enter. The actual information to be read by your interface program are contained in the lines without the '#' symbol. The following are the setup information contained in the file:

- Type of the problem to be solved
- Name of the file containing the specifics of the problem
- Type of the search algorithm to be used
- Step size of axis-parallel mutation for numerical optimization
- Number of consecutive iterations with no improvement (for the first choice and stochastic)
- Update rate for the gradient update rule
- Size of increment to be used for calculating derivatives
- Number of random restarts (for hill climbers)
- Total number of iterations until termination (for metaheuristic algorithms)
- Total number of experiments to be conducted

Changes to the Main Program

The functions in the main program should all be rewritten to support the new requirements. There must be functions that read setup information from a file and then create appropriate objects. There also should be a function that works as a wrapper around the selected optimizer to conduct multiple experiments. The wrapper function should be able to run the selected algorithm multiple times and collect the intermediate and final re-

sults. You are provided with the new main program with a couple of functions left incomplete. Your task is to complete the program by filling up the codes for those functions (indicated by '###' after the function name).

Changes to the Class Hierarchy

New variables and accordingly new methods should be added to various classes because of the new interface. Some variables may have to be relocated to a different level within the hierarchy. A general principle is that a variable *v* is better placed at the lowest possible class *C* such that *v* is used in that class or in its subclasses. There are especially many variables that need to be added to the 'Problem' class for storing the results of experiments:

- Name of the file containing problem specifics (this is actually not a result but a setup information)
- Best solution found all through different experiments
- Its objective value
- Average objective value of the best solutions obtained from *n* experiments
- Average number of evaluations made in *n* different experiments
- Average iteration when the best solution first appears in *n* experiments (only for metaheuristic algorithms)
- Total number of evaluations made all through *n* experiments

Accordingly, the codes for reporting the results need to be revised significantly.

Note that metaheuristic algorithms, unlike hill climbers, are run for a predetermined number of iterations, which is actually not easy to be determined *a priori* and may be determined only by trial and error. Therefore, after a run, one may want to know exactly when the best solution found actually appeared the first during the iterations. The sixth item listed above is the average of this iteration when experiments are conducted multiple times.

Some Useful Codes for Implementing Stochastic Hill Climbing and Simulated Annealing

You are provided with three methods, 'stochasticBest', 'initTemp', and 'tSchedule', which can be used when implementing stochastic hill climbing and simulated annealing algorithms.

- Stochastic hill climbing generates multiple neighbors and then selects one from them at random by a probability proportional to the quality. 'stochasticBest' can be used for this purpose.
- Both 'initTemp' and 'tSchedule' will be called by simulated annealing. 'initTemp' returns an initial temperature such that the probability of accepting a worse neighbor is 0.5, i.e., $\exp(-dE/t) = 0.5$. 'tSchedule' returns the next temperature according to an annealing schedule.

Plotting the Progress of Search by Using Matplotlib

Solve the TSP in the file 'tsp100.txt' by using both the first-choice hill climbing and simulated annealing. You should modify your program so that the objective value of the current point at every iteration is recorded in a text file. Then, use the text files to plot and compare how the value changes differently with each algorithm from the beginning to the end of the search.