Optimization Mini-Project Definition:
Enhancing the simple hill-climber

In this mini-project, we are going to enhance the simple hill-climber from Lesson 2, to be able to find the global optima of a few test-problems. In order to do this, the following assignments shows how to improve the given simple hill-climber gradually. The following assumes that we implement the changes using Java.

Handout 1 (General Problem class):

In order to represent the two test-functions, we first need a general Problem class, which we can extend to the concrete test problems. This is created as an abstract class. The problem class has the following attributes:

1. An ArrayList of doubles called maxValues, representing the maximum values for each of the problem parameters.
2. An ArrayList of doubles called minValues, representing the minimum values for each of the problem parameters

Further, the Problem class has methods to get and set these, as well as a constructor, that sets the initial two ArrayLists to be empty lists.

Further, the class has an abstract helper function getDimensions that takes no argument, but returns the number of dimensions for the problem, that is, the number of parameters.

Lastly, the class features an abstract method called Eval, which takes an ArrayList of parameters, and returns the evaluation function value as a double.

Handout 2 (Concrete Problem classes):

Further handed out is three concrete extensions of the Problem class, called P1, P2 and RevAckley. In the constructors of P1 and P2, the maximum and minimum parameter values are set as follows:

1. For P1 and P2, the maximum value for both parameters (x and y) is 2.0.
2. For P1 and P2, the minimum value for both parameters (x and y) is -2.0.

Further, P1 and P2, the Eval and getDimensions methods are overridden as follows:

1. For P1, the Eval function must return e^-($x^2 + y^2$), with e being the mathematical constant e, and the ^ sign denoting the power function.
2. For P1, the getDimensions function returns 2.

P1 is the test function presented in Lesson 2, on slide 3.

1. For P2, the Eval function must return e^-($x^2 + y^2$) + 2e^-($(x – 1.7)^2 + (y – 1.7)^2$), with e being the mathematical constant e, and the ^ sign denoting the power function.
2. For P2, the getDimensions function returns 2.

P2 is the test function presented in Lesson 2, on slide 5 (to the left).

RevAckley is the reversed version of the Ackley test function (in order to maximize it), that can be found on Wikipedia: https://en.wikipedia.org/wiki/Test_functions_for_optimization.

The Eval and getDimensions methods are overridden accordingly.

Assignment 1 (Using the Problem class)

Prepare the hill – climber for taking in a problem to solve, by creating an instance variable of Problem type, that is to be initialised by the constructor. Alter the constructor as to allow a Problem instance as an argument. This enables us to use the quality of a solution as calculated using the Eval function of the current Problem, in the hill – climbing process. It is optional to use a Strategy pattern, to be able to change the current problem.

Assignment 2 (Iterating the hill - climber):

Next, we do our first improvement on the hill – climber, by making it iterative, so that it may restart a number of times, and at the same time always remembering the best solution found so far.

Then, in order to be able to iterate our hill – climber and being able to return a point instead of a single value, we change the findOptima methods as follows:

1. Change the return type to be an ArrayList of doubles.
2. Change the number of arguments to findOptima to 1, an integer specifying the number of iterations, called *iterations*.

Next, create a loop that runs the hill – climbing part of the findOptima method *iterations* number of times, each time starting from a random solution of the same dimension as the current problem. Remember the global best solution found in all iterations, and return this in the end of the run.

The following Java – snippet shows one possible way of creating the random initial point (within the maximum and minimum values), with curProblem being the current problem:

```
ArrayList<Double> initPoint = new ArrayList<>();
for (int dim = 0; dim < curProblem.getDimensions(); dim++) {
       initPoint.add(curProblem.getMinValues().get(dim) + Math.random() *
       (curProblem.getMaxValues().get(dim) - curProblem.getMinVal-
       ues().get(dim)));
       }
```

Assignment 3 (Neighbour creation):

Further, we need to create a new neighbour solution. We can do this several ways, one being subtracting or adding (a maximum of) a small value, *stepsize,* to each parameter value of the current locally best solution with a 50/50 probability. We can ensure that the change is at max *stepsize* by adding/subtracting Math.random() * *stepsize* to each parameter. Note, that the current locally best is initially the random solution. Further, check if the new parameter value is violating the min and max values, and set it to the border value, if it is. Add the *stepsize* parameter to the findOptima method.

Assignment 4 (Sampling the neighbourhood):

Next, we need to generate several neighbours, and then select the best one as a basis for the continued hill – climb (if it is better than the current locally best). Do this by adding one more parameter to the findOptima method, specifying the number of neighbours generated, called *neighbours*. Further, create a loop around the creation of a neighbour, so that *neighbour* new solutions are created, and so that the outer loop continues with the best of these.

At the end of the hill-climb, test the local best against the global best, and if the local best is better, save this as the new global best solution.

Assignment 5 (Testing and reporting):

The three concrete problem classes are the subject of testing with your final version of the iterated hill – climber. The overall goal is to find the global optima of the three problems, by varying the parameters of the hill – climber.

Hand – in your final version of the hill – climber as a .jar or .zip file (if anything other than Java is used) as well as a small report (max five pages, including figures, e.g. graphs) on Canvas.

The testing consist of varying the parameters of the hill – climber in order to find the global maximum. Conduct the following experiments:

1. Vary the *iterations* parameter – try 10, 100 for P1 and P2 and further 200 for RevAckley.
2. Vary the *neighbours* parameter – try 10, 100 for P1 and P2 and further 200 for RevAckley.
3. Vary the *stepsize* parameter. Try 0.1 and 0.01 for P1 and P2 and further 0.001 for RevAckley.

For each of your experiments, include the number of new solutions generated in the run of the hill – climber (by counting each time a new solution is created, as they each need to be evaluated).

Further, during experimentation, print or log the global best function value found in each iteration, to see when there is an improvement. Also, try to print out each time a better solution was found during the neighbourhood sampling.

Report and reflect on the following:

1. Your experiments in general.
2. Where do you think improvement happens in the iterated hill – climber?
3. The effectiveness of the experiments concerning the number of new solutions created.
    a. Can you explain the numbers?
4. Based on your experiments, what do you think the global optima for the functions are?
5. Give both the best solutions found and the evaluation function value of these.
6. How close do you think you came to the global optima? Can you find them analytically?
7. What was the best parameter setting for your iterated hill – climber, that is the best settings (giving the best results considering the number of evaluations) of *iterations*, *neighbours* and *stepsize* for each problem?

Deadline for the hand-in is Thursday, September 17, at 16.00.

Peter Justesen

Project Definition