CPSC 2120/2121

Lab 08

Hill Climber and Simulated Annealing
Due date/time are on the CPSC 2121 Canvas website

Learning Objectives

- To improve our ability to implement algorithms and data structures in C++
- To implement a heuristic search in C++
- To use simple data structures to implement algorithms
- To become proficient in fundamental data structures used throughout computer science
- To improve our analytical skills while gaining familiarity with mathematical tools used in the analysis of algorithms
- To implement an algorithm and understand its intricacies

Problem/Exercise

For this lab, you will write two closely related optimization functions: a hill climber and simulated annealing. For each function, you must find the minimum possible output. Give the input and output for that minimum output. You will be required to do this for each function given below.

Recall from class that at each iteration, a greedy algorithm searches the immediate area for the direction of maximum gain and takes one step in that direction. This continues until no direction offers improvement beyond the threshold, which terminates the program. The size of each step may be standard or you may generate a random step size for each step.

For simulated annealing, you will use the heat function P(x,T) = e^((old - new)/T)

old = current value
new = random neighbor near the current value
T = current temperature

You are allowed to use any data structures in the C++ STL for this assignment. Your final output will be in a single file, named optimizers.h, which will compile without errors using the provided Makefile. In order to thoroughly test your program, you will also submit a single input file named tests.h which has map information and queries in the same format as the provided tests.h. It will be necessary to add additional test cases to the tests.h file, following the provided example.

Additional Requirements

- Your optimizers.h file must compile and run with unmodified versions of the provided tests.h and Makefile to produce the same output as our examples. If your class does not, then you'll need to fix it. After you get your class working, you should create several examples on your own to further test your program to make sure it works with any valid set of test cases.
- Your code must run for each of the following functions. The last two are on this wikipedia page. Domains for all functions are real numbers.
    - $x^2 + (x - 2)^2$
        - domain: $-100 \leq x \leq 100$
    - Schaffer function N.2
        - domain: $-100 \leq x,y \leq 100$
    - Bukin
        - domain:
            - $-15 \leq x \leq -5$
            - $-3 \leq y \leq 3$

Examples

Your C++ source code should be in a file called optimizers.h, and it should be compiled into an executable called lab08.out using our provided Makefile. The output of optimizers.h must look exactly like the examples below when run on the command line on our Unix machines.

./lab08.out
x^2 + (x-2)^2
Greedy: min = 2, x = 1
SA: min = 2, x = 1

schaffer
Greedy: min = 0, x = 0, y = 0
SA: min = 0, x = 0, y = 0

bukin
Greedy: min = 0, x = -10, y = 1
SA: min = 0, x = -10, y = 1

Source Code Requirements

- Put a comment at the top of your source code file with your name (first and last), the date of your submission, your lab section, and the assignment's name.
- All functions should be commented. Use inline documentation as needed to explain ambiguous, tricky, or important parts of your code.
- All source code must follow good programming style standards such as properly indenting source code; and all variables, functions, and classes should be well-named.
- Your class must use dynamic memory allocation and deallocation properly, which means your class cannot contain a memory leak. You may use valgrind to check for memory leaks. Refer to the Unix manual and valgrind's online documentation for more information on valgrind.

Submission

Before the date/time stated on the CPSC 2121 Canvas webpage, you need to submit your code to handin under the correct lab assignment. Make sure to submit all of the following.

1. All source files required for this lab (optimizers.h)
2. Your testing file (tests.h)

After you submit, always double check that the file(s) you submitted were the correct version. To double check, download the submitted file(s), put them on one of our Unix machines, and make sure they compile and run correctly.

Grading

If your class does not compile on our Unix machines or your assignment was not submitted on time, you will receive a grade of 0 on this assignment. Otherwise, your class will be graded using the criteria below.

Your class works correctly with our testing program(s)	6 points
Proper variable names, documentation, and code organization	2 points
Your class uses dynamic memory allocation/deallocation properly (no memory leaks)	1 point
Alternate testing file containing map and query data you created for testing which is distinct from that provided.	1 point

You must test, test, and retest your code to make sure it compiles and runs correctly and efficiently on our Unix machines with any given set of valid inputs. This means that you need to

create many examples on your own (that are different from the provided tests.h) to ensure you have a correctly working class. We will only test your program with valid sets of inputs.