Why?

The purpose of this project is to explore random search. As always, it is important to realize that understanding an algorithm or technique requires more than reading about that algorithm or even implementing it. One should actually have experience seeing how it behaves under a variety of circumstances.

As such, you will be asked to implement or steal several randomized search algorithms. In addition, you will be asked to exercise your creativity in coming up with problems that exercise the strengths of each.

As always, you may program in any language that you wish insofar as you feel the need to program. As always, *it is your responsibility* to make sure that we can actually recreate your narrative if necessary.

Read everything below carefully!

The Problems You Give Us

You must implement four local random search algorithms. They are:

1. randomized hill climbing
2. simulated annealing
3. a genetic algorithm
4. MIMIC

You will then create (for sufficiently loose values of "create" including "steal", though it's fairly easy to come up with simple problems on your own in this case) three optimization problem domains. For the purpose of this assignment an "optimization problem" is just a *fitness function* one is trying to *maximize* (as opposed to a cost function one is trying to minimize). This choice doesn't make things easier or harder, but picking one over the other makes things easier for us to grade.

Please note that *the problems you create should be over discrete-valued parameter spaces. Bit strings are preferable.*

You will apply all four search techniques to these three optimization problems. The first problem should highlight advantages of your genetic algorithm, the second of simulated annealing, and the third of MIMIC. Be creative and thoughtful. It is not required that the problems be complicated or painful. They can be simple. For example, the 4-peaks and k-color problems are rather straightforward, but illustrate relative strengths rather neatly.

The Problems Given to You

In addition to analyzing discrete optimization problems, you will also use the first three algorithms to find good weights for a neural network. In particular, you will use them

instead of backprop for the neural network you used in assignment #1 on at least one of the problems you created for assignment #1. Notice that this assignment is about an optimization problem and about supervised learning problems. That probably means that looking at only the loss or only the accuracy won't tell you the whole story. Luckily, you have already learned how to write an analysis on optimization problems and on supervised learning problems; now you just have to integrate your knowledge.

Because we are nice, we will also let you know about some pitfalls you might run into:

- The weights in a neural network are continuous and real-valued instead of discrete so you might want to think a little bit about what it means to apply these sorts of algorithms in such a domain.
- There are different loss and activation functions for NNs. If you use different libraries across your assignments, you need to make sure those are the same. For example, if you used scikit-learn and don't modify the ABAGAIL example, they are not.

What to Turn In

You must submit:

1. A file named *README.txt* that contains instructions for running your code and some way of getting to your code, just like last time.
2. a file named *analysis.pdf* that contains your writeup.

The file *analysis*.pdf should contain:

- the results you obtained running the algorithms on the networks: why did you get the results you did? what sort of changes might you make to each of those algorithms to improve performance? Feel free to include any supporting graphs or tables. And by "feel free to", of course, I mean "do".
- a description of your optimization problems, and why you feel that they are interesting and exercise the strengths and weaknesses of each approach. Think hard about this.
- analyses of your results. Beyond answering why you got the results you did you should compare and contrast the different algorithms. How fast were they in terms of wall clock time? Iterations? Which algorithm performed best? How do you define best? Be creative and think of as many questions you can, and as many answers as you can. You know the drill.

Note: Analysis writeup is limited to 10 pages total.

Coding Resources

Just like in project 1, you can use any library as long as it wasn't specifically written for this class, particularly for automating away your personal analysis. While looking at libraries, you might want to take a look at ABAGAIL:

https://github.com/pushkar/ABAGAILLinks to an external site.

Grading Criteria

At this point you are not surprised to read that you are being graded on your analysis more than anything else. On the other hand, I will also point out that implementing some of these algorithms is very easy (almost not worth stealing the code, but please feel free to do so anyway (I would steal the code)) but at least one of them requires some time (luckily, there are now versions of this algorithm out there to steal).