CS 4100/5100 – Foundations of Artificial Intelligence

Assignment 4: Genetic Algorithms[1]

Due: November 21st, 11:59pm
Note that I am providing two extra days for this assignment: this is not so that you can take a mini-vacation from homework! Genetic algorithms can take a very long time to run, and I want to make sure you have plenty of time to get the code written and create your report.

Learning Objectives

- Design and implement a simple genetic algorithm with crossover, mutation, and elitism
- Experiment with different representations and algorithm parameters

Assignment Description

For this assignment, you will be writing your own genetic algorithm that can learn to mimic an arbitrary source image: I am supplying you with three example images, but feel free to use your own, too! We reserve the right to grade the project on an image of our choice.

There are four main components to building a genetic algorithm:

1. Determining a representation for your population to use
2. Defining an appropriate fitness function
3. Authoring the genetic algorithm itself: creating a population and evolving it using crossover, mutation, and elitism
4. Comparing the results of your work when altering different aspects of the algorithm

The Representation

The sample code provides a *Circle* class that represents circles as a point in 2D space ($xPos, yPos$), a radius ($radius$), and a color (r, g, b, a). The provided Candidate class uses the representation of an array of *Circle*s. These circles and their parameters form the DNA that will be evolved.

Graduate students: you are required to implement one additional representation (suggestions: ellipses, rectangles, lines, n-sided polygons, fonts). Undergraduate students: you may use the representation provided.

---

[1] This assignment is based on a project that originally (as best I can tell) was done by Roger Alsing: http://rogeralsing.com/2008/12/07/genetic-programming-evolution-of-mona-lisa/

Note that Alsing's project is not actually using a genetic algorithm as his population size is one – as best I can tell from the information given, his approach is a hill-climbing optimization search. Let's see how it works with genetic algorithms instead.

Please be aware that I know there is code on the internet that does something similar to what I'm asking you to do, and I know what it looks like. Using it as a reference is fine, but you **must** state in your README that you did so and give credit **in your code** to what you are re-using or basing your work on.

The Fitness Function

The fitness function tells how well the candidates in your population are meeting the desired goal. For this assignment, you must define a fitness function that says how well your candidate population is matching the provided image. *Candidate* has a method, *getCandidatePixels()*, that returns a single-dimensional array of pixels corresponding to what the candidate looks like when drawn on the screen. There are global helper functions – *getRed(color c)*, *getGreen(color c)*, *getBlue(color c)* – that will return the red, green, and blue components of a pixel's color as an integer on [0, 255]. The variable *goalPixels* refers to the array of pixels associated with the goal image that you load in.

All students must implement a fitness function that judges how well a candidate in the population matches the goal image. You could do this just by comparing each pixel value to see how far away the candidate is from the goal per pixel, or do something different!

The *Candidate* class has a stubbed function, *calculateFitness()*, that you should fill out. You can change the method signature if necessary, it's just there as a starter.

You may choose to implement an additional fitness function as part of your experiment (see below).

The Genetic Algorithm

You are required to implement a genetic algorithm that can do the following:

- Crossover two candidates to produce a new one
- Mutate a single candidate
- Optionally keep the best x% of candidates

There are stubbed methods in Candidate for crossover and mutation. There are variables at the top of the main file that should control your algorithm somehow. Feel free to add new variables to control additional parameters.

You may choose to implement additional crossover and mutation methods as part of your experiment (see next section).

The Experiment

Altering the representation, fitness function, and parameters to your genetic algorithm can drastically impact its performance. In addition to your code, you are required to submit a written report showing examples of the output of your system using different algorithm parameters and representations at different phases of generation.

You should show images that are produced at a minimum of 5 different iterations of the algorithm: the first iteration, the last (where you have either reached an optimum or come as close as you can get), and three stages in between. You might find the built-in Processing function *save(String filename)* useful to automatically save your images during generation.

You should then show images at the same stages of the algorithm but with different algorithm parameters, representations, or fitness functions. You should do this for at least four different versions of your program; graduate students are required to show both representations that were implemented in the written report. When showing how the outcome varies for different parameters, write a couple of sentences about why it is you think the outcomes occurred based on the parameters, representations, and fitness functions used. Does your algorithm work better for some images than for others?

In summary, to receive full credit for the report, you must:

- Show the results of four different versions of your algorithm, where each version uses different generation parameters
- Graduate students must include in the report one version of the algorithm that uses the second implemented representation
- Write a few sentences about why you think the algorithm is performing differently (or similarly) when changing these parameters

Extra credit may be given for students who implement additional features (e.g. multiple fitness functions, representations, substantially different crossover and mutation methods, more sophisticated genetic algorithm features). There is a maximum of 20 points of extra credit that can be given on this assignment.

Supplemental Files

You are provided with code written in Processing that will handle the graphics portion of this assignment for you: drawing circles to the screen, reading in an image file, rendering your population to a frame buffer, and getting individual pixel values. Processing is Java, but with lots of built-in functionality that makes it easy to create interactive visual environments, and a simplified UI.

I encourage you to use the provided sample code so that you can focus on the AI portion of the assignment, but if you are more comfortable starting from scratch in either Java or C++, that is fine! Also, Processing can be used in a more fully-featured IDE (e.g. Eclipse, IntelliJ, NetBeans) with some small amount of configuration effort. You can find tutorials online for how to do this.

As long as your assignment can be run on one of the lab machines in 102 and you provide plenty of documentation, it is gradable.

Here are the supplemental files I am providing:

- evo_photo_base.zip – a zip file containing the Processing sample code, which itself is split into three files and a data directory:
    - evo_photo_base.pde – the main Processing file
    - candidate.pde – the Processing file containing the Candidate class
    - shapes.pde – the Processing file containing the Circle class
    - data/ -- a directory containing three sample image files

Submission Instructions

You are required to submit a zip file containing:

- a .zip of your Processing project
- documentation stating how to run your project – any variables that can be changed, how to run it using different representations, etc.
- a PDF or Word document containing your project report
- a readme text file with your name, the number of late days you wish to apply, and the names of any people or resources you used in creating your project

Your zip file should be uploaded by November 21st at 11:59pm. All materials must be submitted via Blackboard. **Assignments emailed to me will not be accepted.**