CS4341

Assignment #2: Genetic algorithms

Due date: February 14, 2022 @ 1:59 pm

This assignment will familiarize you with genetic algorithms (GA). You will solve a variety of problems to learn about representation (the biggest challenge for GA). You will also conduct experiments **with a couple of variants of GA to see how they affect performance.**

Framework

It is recommended that you first develop a codebase for how your GA will work at a high level. In particular, you need to determine:

1. How large is the population?
2. Fitness function: how members of the population will be judged on the current task. Using the score on the task (provided below) is certainly one approach, but you are free to use a different fitness function for selection.
3. A selection rule: how will members of the population be selected to continue in the next generation?
4. Crossover: how will your algorithm decide where to split parents for offspring?
5. How often should mutation occur ("never" is a possible answer, but may limit your agent's performance)? How should mutation be implemented?
6. Anytime performance: your algorithm should take as input a particular length of time (in seconds), run for that duration, and output the best result it has found so far. For this assignment, it is acceptable to check the time at the start of each generation and finish a generation that starts before time expires.
7. **For both puzzles, higher scores are better (added 2/7 @ 2:06 pm)**

Puzzle #1: number allocation

In this puzzle, you will be given a series of 40 numbers, one per line. Your agent's job is to allocate each number into 1 of 4 bins:

- Bin #1: the numbers are all multiplied together

- Bin #2: the numbers are all added together
- Bin #3: the smallest number is subtracted from the largest
- Bin #4: these numbers are ignored

Each bin must have 10 numbers allocated (i.e., the same number in each bin). For this task, numbers will be floating point in the range [-10, 10]

The score for the task is the summed score of bins #1, #2, and #3.

An example with 8 numbers (for brevity):

-6.3
2.5
8.8
0.6
-4.5
2.3
9
1.1

With an allocation of:

- Bin #1: -6.3, -4.5
- Bin #2: 8.8, 2.5
- Bin #3: 1.1, 9
- Bin #4: 0.6, 2.3

Would score:

- Bin #1: (-6.3 * -4.5) = 28.35
- Bin #2: 8.8 + 2.5 = 11.3
- Bin #3: 9 - 1.1 = 7.9
- Overall score = 28.35 + 11.3 + 7.9 = 47.55

Your program should output the numbers in each bin, with one line per bin. For example:

-6.3 -4.5
8.8 2.5
9 1.1
0.6 2.3

Puzzle #2: tower building

Towers are composed of pieces. Each piece is represented in one line (tab delimited). Pieces have the following properties:

1. Type: either door, wall, or lookout (String)
2. Width: how wide the piece is (Natural)
3. Strength: how many pieces can be stacked on top of it (Natural)
4. Cost: how expensive it is to use this piece in a tower (Natural)

Each piece may be used at most once. Towers are composed of a collection of pieces stacked on top of each other. **Useful** (ADDED THE WORD "useful" FOR CLARIFICATION) Towers must obey the physics of the world:

1. Towers **must** have a door as their bottom-most piece.
2. The top piece in a tower **must** be a lookout.
3. Pieces between the top and bottom of a tower, if any, must be wall segments. I.e., towers can only have one door and one lookout.
4. A piece in a tower can, at most, be as wide as the piece below it.
5. A piece in a tower can support its strength value in pieces placed *above* it.

Towers that do not follow those rules are worthless as they will either collapse or are not usable by their future owners. Unfortunately, your GA does not know the physics of the world ahead of time or people's preferences. It must build a tower to see that that tower has a fitness of 0. Therefore, you should **not** use those rules for the seeding the initial population. Part of this problem is for your agent to discover the rules of the world.

The score of a tower is ($10+height^2$ – piece cost to build the tower). A tower that violates any of the above 5 constraints receives 0 points. A tower of height 2 with a door on the bottom and lookout on the top is a legal tower.

For example, given the following pieces

- Door, 5, 3, 2
- Wall, 5, 5, 1
- Wall, 4, 3, 1
- Door, 3, 5, 2
- Wall, 3, 3, 2
- Lookout, 2, 2, 3
- Lookout, 3, 1, 2

If your agent constructed the following tower:

Door, 5, 3, 2
Wall, 5, 5, 1
Wall, 4, 3, 1
Lookout, 3, 1, 2

It would be of height 4, and would score 10 + 16 – (2+1+1+2) = 20 points.

The following tower:

Door, 5, 3, 2
Wall, 4, 3, 1
Wall, 5, 5, 1
Wall, 3, 3, 2
Lookout, 2, 2, 3

Would score 0 points as it has multiple problems. First, the second wall segment is wider than the first wall segment, a violation of the rule #4. Second, the door has a strength of 3, but has 4 pieces placed on top of it, a violation of rule #5.

Your program should output the towers as shown above: one piece per line with all of the information.

Requirements

Part of this assignment is for you to observe your GA learning the rules of the puzzle. Therefore, you should not seed your initial population with what you think are good guesses. For example, for puzzle #2, you should not randomly generate towers with a door at the bottom, one or more valid walls, and a lookout on top. The towers should be random. An exception for puzzle #1: you may restrict initial members of the population (and crossovers) to having 10 numbers in each bin, as that is a constraint on the puzzle that would be very difficult for the GA to learn.

For mutation, you cannot change the starting values. I.e., you cannot create new numbers or new tower pieces. Mutation could swap numbers between bins, or swap around the location of tower pieces.

Violation of this protocol **will lose you points**.

Code behavior

You should hand in a program called ga that takes 3 command-line arguments:

1. Which puzzle it is solving (1 or 2)
2. The name of the file it should read in that contains the puzzle information.
3. How many seconds it has to work on a solution.

Your code should also not have a preset random number seed when you turn it in (if you don't know what this sentence means, you're fine—you have to take explicit steps to give it a specific random seed). That way if we rerun your code the behavior should vary a bit.

Part of the goal for this assignment is to get you to consider the impact of different design decisions on your agent's performance. Therefore, having slightly different performance each time is good.

Your code should output the best solution it found, its score, how many generations it ran for, and at what generation it found its best solution.

Writeup

You must include a writeup that explains the following aspects of your code:

1. Your approach to computing a fitness function, selection, child generation, crossover, and mutation. Also, what population size did you use? You may use the same approach to these for both puzzles, but are not required to. Provide an example of evaluating a small population, selection, and generating a child from one set of parents. If your approach varies across problems, show an example of each.
2. How you handled the problem of illegal children for crossovers. Provide an example, or explain how your representation prevented the problem from occurring.
3. For both puzzles, provide a graph showing how solution quality varies as the number of generations increases:
    a. On the x-axis, plot the number of generations. If you want, you can sample, and only examine performance every so often (e.g., every 100 generations)
    b. On the y-axis, plot performance of your algorithm. Specifically, plot the best performance of the population, the worst performance, and median (50^{th} percentile) performance.
    c. As GAs are non-deterministic, you should run your code 5 times on each puzzle, and report the average result on each of the metrics when

generating the graphs. So you will have one graph that summarizes 5 runs of your agent on a puzzle.

4. You must also experiment with a few simple techniques with GA and explain how they impact performance on the 3 puzzles. In addition to your baseline performance, you should generate graphs like the ones described in step #3, with and without:
    a. Elitism: experiment with retaining the top-performing members of the population on each generation.
    b. Culling: experiment with dropping the bottom-performing members of the population on each generation.
    c. Explain what effect these two techniques have on the 3 metrics (best-, worst-, and median-performance)

You should explain how you implemented elitism and culling, and how you settled on your approach.

The writeup should be done with a word processor and submitted as a .docx or .pdf file.

What you should hand in: a zip file containing

1. Your program. Include any instructions for how to execute the code.
2. A test file for each puzzle that has your algorithm asymptoting in performance after about 10 seconds. These files are the ones you should use in conducting the experiments in your writeup.
3. Your writeup.

Grading guidelines

Correctness: 55%. Have you implemented the components of genetic algorithms correctly (e.g., selection, crossover, mutation)? Does your program create a viable GA that improves over time? Are the solutions it claims are valid actually valid?

Performance: 15%. We will judge the quality of your agent's solutions on the two puzzles, using a standardized set of boards for grading. Allocating time efficiently is key for local search (and for many problems in AI); experiment to find out what works best.

Writeup: 30%. You should explain what you did in your code, and report on the experiments you ran investigating performance of your GA. Clarity and examples are far more important than many pages of text.

Both puzzles have equal weight in the grading.

Hints

1. This assignment is longer than assignment #1, so get started on it immediately.
2. Get puzzle 1 working, and think about how you can write your code generally enough so that it works for puzzle 2 as well.
3. For puzzle #2, it may be helpful to have a representation that has three components: bottom, top, and everything in between.
4. There are a lot of non-correctness points for this assignment. Don't wait until the last minute to think about optimizing your GA's hyperparameters and getting the writeup done.