Project 2: Adding some functionality

Due: March 17th, 2022

Total: 100 points.

Invocation (same as previous project):
> sga [-h] [-g] [-G] [filename]
with filename being the name of your settings file. If this filename is left off, by default it should use a settings file called "gasettings.dat". All of the arguments are optional

-h : Should output a help message describing what the options do and then terminate..

-g : Limited debugging information should be displayed while running.

-G : Full debugging should be displayed while running.

The settings file is same as before, with additional options described in this project

Task 1: Add two new crossover operators:

You must add one-point and two-point crossover to your GA. One-point crossover, as described in class, picks one random crossover point in our strings and swaps the resulting parts from each string which results in two (possibly) different offspring. Two-point crossover picks two random crossover points and swaps the parts in alteration to generate, again, two possibly different offspring.

For example, for one-point crossover suppose we have a string that is split at some particular point, such that all the parts of the strings before the crossover point are in A and all the points afterwards are in B. Then we have another string with all the points before the split point are C and all after are D. Our two strings, AB and CD, if crossed over, would result in strings composed of AD and BC.

For two-point crossover, it is similar. Again, suppose we had a string split at two points, represented by ABC. Another string, split at the same two random points, DEF. Then the resulting offspring would be composed of each parts of the string as AEC and DBF.

You should also add the following options to your settings file.

crossoverOperator x

where crossoverOperator 1 would use one-point crossover and crossoverOperator 2 would use two-point crossover. crossoverOperator 0 should use uniform crossover as per our previous projects.

Task 2: Add a new fitness function

Our new fitness function is called trap-4. As in our previous fitness function of onemax, the global optimum is the string of all 1s. However, it varies quite a bit on values that are not that. Keep in mind also that this fitness function expects strings of size divisible evenly by 4.

In particular, trap-4 partitions the problems into groups of 4 bits. The first 4 bits of the chromosome are part of partition 1. The second group of 4 bits make up partition 2 and so on. The overall fitness score is the sum of the fitness of each partition. So, how is the fitness of each partition calculated? It depends on how many 1's are in each partition.

If the number of 1s in its partition of 4 bits are:

0 1's : The fitness score of the partition is 3

1 1's: The fitness score of the partition is 2

2 1's: The fitness score of the partition is 1

3 1's: The fitness score of the partition is 0

4 1's: The fitness score of the partition is 4

For example, if a partition was "0100" it's individual partition fitness would be 2. This would be true for a partition of "1000" or "0010" or "0001".

You would then tally up all the fitness scores of each partition and that would be your overall fitness.

This fitness function should be used if the setting:

fitnessFunction 1

is set.

Task 3: Add a different way to run your GA (Bisection)

We often want to figure out what the minimum population is to solve a particular problem. The so-called “Bisection” method can help us do this. The bisection method is a procedure to run your GA using various population sizes in order to automatically “narrow down” to the minimum population required to solve a problem.

In bisection we start off with some small N, let us say N=10 for our population size and try to solve some problem before 50 generations have passed. We then run our GA for those settings. It will almost surely fail, as our population is so small. Then we double it, so N=20 and try again. If it fails, we repeat again, for N=40. We repeat this process until we succeed.

At this point, let us say we succeeded at N=160 but failed at N=80. We then try and narrow this down by searching our space to see what the smallest population size is such that we still succeed. To do this, we split our space up by averaging the failure size and success to get (80+160)/2 = 120. We then try 120. If we fail, we then split between our failure of 120 and success at 160 by trying (120+160)/2 = 140. If we succeeded at 120, we split between our failure at 80 and our success at 120 by (120+80)/2 = 100. We repeat this process until we get close enough in our bounds.

The pseudocode of the method is as follows:

```
1. Start with some very small N (say, N=10)

2. Double N until GA convergence is sufficiently reliable

3. (min,max)=(N/2,N)

4. repeat until (max-min)/min<0.1 (use your own threshold here)

    N=(min+max)/2

    if N leads to sufficiently reliable convergence

      then max=N

      else min=N

5. Compute all your statistics for this problem size using

  pop. size=max
```

You should include an option in your settings file “bisection 1” if you wish to use bisection. bisection 0 would just do one individual run.

After running under bisection, your algorithm should output what the final result of N is.

TASK 4: Experiments

Experiments:
I want you to experiment with the GA in order to see what operators work well and what do not for onemax and trap-4. To do this, use bisection to get an idea of what the minimum population size is required to solve under the given settings.

In particular, I want you to test trap-4 and onemax on strings of size 24 with the goal of solving them before 100 generations using bisection. Then do the same for string sizes of 48 before 200 generations. Lastly, try it for string sizes of 100 before 300 generations.

What parameters should you vary? You should try all our different crossover operators (uniform, two-point, one-point).

Submission:
I expect the following:

1) A readme.txt file describing how to compile and run your project. This file should also outline briefly the results of your experiments above.

2) A word document or pdf (called results.pdf or appropriate suffix) describing your experiments and results of bisection on all the different settings variants.

3) A summary at the end of this word document detailing any conclusions you can draw from this.

4) All your source files

To submit, put all these files in a folder and compress them using tar or zip. Then attach the archive to canvas.