Sam Trenter
Project 3

I want you to experiment with the GA in order to see what operators and replacement methods work well and what do not for onemax, trap-4 and interleaved trap4. To do this, use bisection to get an idea of what the minimum population size is required to solve under the given settings

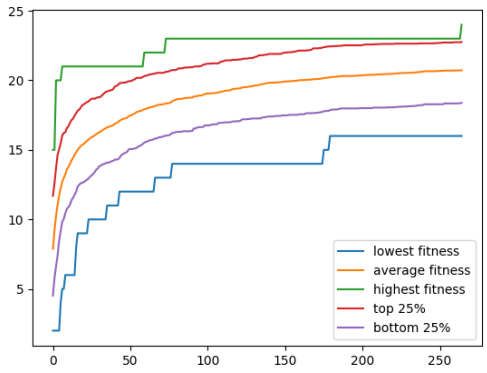
So for Trap 4, one and two point crossover worked well, for Interleaved Trap4 both one and two point crossover worked terribly. The only thing I’ve found to work well is uniform crossover. This makes sense because you are basically hoping that each of the 4 bits you need get flipped. You can actually notice this during the running of the GA as for the last few Generations, only one section won’t have the correct bits. IE: notice how only one section doesn’t have the correct bits, meaning the individual is only one point away.

```
Generation:  638
Most Fit
=-=-=-=-=-=-=-=-=-=-==-=-=-=-=-=-=-=-=-=
fit:  31
11101111111011111110111111101111
=-=-=-=-=-=-=-=-=-=-==-=-=-=-=-=-=-=-=-=
least fit
=-=-=-=-=-=-=-=-=-=-==-=-=-=-=-=-=-=-=-=
fit:  20
00110001000100000111000000110000
=-=-=-=-=-=-=-=-=-=-==-=-=-=-=-=-=-=-=-=
avg fit:  28.13
Generation:  639
Generation:  640
Generation:  641
Generation:  642
Generation:  643
Generation:  644
Generation:  645
Generation:  646
Generation:  647
best individual at end of simulation
=-=-=-=-=-=-=-=-=-=-==-=-=-=-=-=-=-=-=-=
fit:  32
11111111111111111111111111111111
=-=-=-=-=-=-=-=-=-=-==-=-=-=-=-=-=-=-=-=
```

I actually had a lot of issues getting this to run with bisection, But I found that a population size of 200 is good for a string size of 24 using UniformCrossover. For one and two point crossover, I would either very quickly get the correct answer or never converge to the right answer, even with a large population.

You can use the same sizes as you did before, so string size of 24, then 48, then 100. If you are having trouble solving at these sizes, try a string size of 12 and 16. If 100 seems trivial, you can scale up from there.

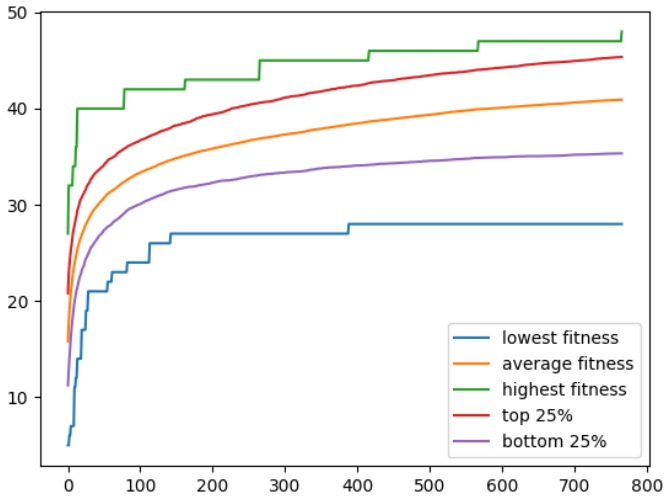
Stringsizen:24



I seem to do an ok job at diversity preservation but I wish I could keep more bad individuals, because in theory they should have more “1”s.

Stringsizen:48

My setup for stringsize 24 didn’t work so I experimented with adjusting settings but I seemed to begin having issues with genetic drift where certain population groups just wouldn’t interact . IE for a 48 bit string I would get all “1”s except for 2 sections that drifted to a zero and would get stuck. They do eventually get unstuck but it takes a very long time.



As you can see the last few trap sections took forever to move up, in theory though this makes sense, because the more trap sections that aren't’ correct, the easier it is to make at least one correct.

Stringsizen:100

This really showed me how slow python and my programming was in general. I couldn't really get this one to compute, and without doing major refactoring making this work on Stringsize 100 would be very time consuming.

What parameters should you vary? You should try all our different crossover operators (uniform, two-point, one-point). You will then want to try for the different replacement methods of elitism and RTR.

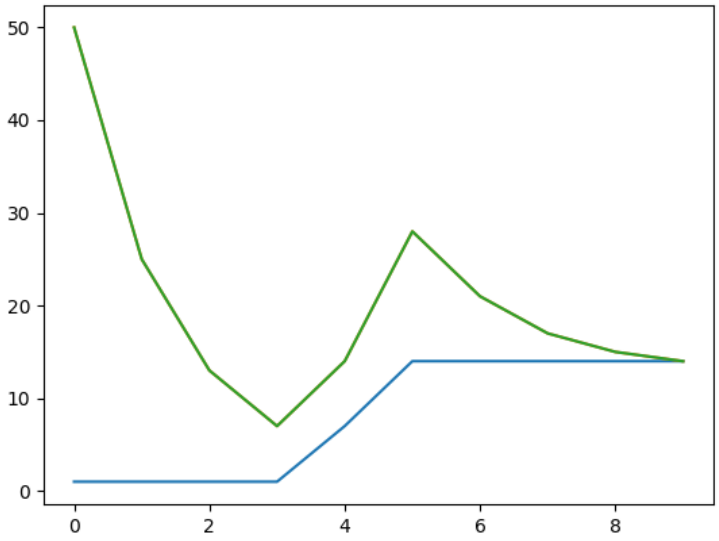
After trying all the crossover operators, one and two point don't work well and uniform crossover works ok. If you think about it, uniform crossover is only better because it has a chance to flip all bits to a one in one go. That's pretty bad.

I've found that changing the "w" size can really affect the results you get. A smaller "w" seems to greatly reduce your divercity but speed up initial learning. While seemingly increasing diversity but slows down learning.

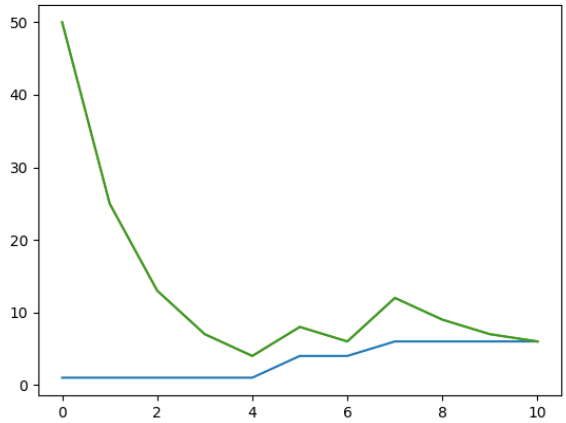
While running slower, RTR usually works better for some settings, but will often fail to converge for most settings.

RTR bisection Stuff:
(I need to refactor to make a ratio and not a integer but It works fine for now)

```
randSeed = 123
populationSizeN = 100
stringSizen = 20        You, 3 hours
probApplyCrossover = 1
probApplyMutation = 1.0
selectionMethod = 0
tournamentSizek = 2
fitnessFunction = 2
crossoverOperator = 0
h = False
g = False
G = False
runmode = 3
bisectionTimeout = 1000
# add below to settingsfile
pauseAtBeginning = False #this just
graphing = True
w = 25 #int(stringSizen * 1)
replacementAmount=populationSizeN
```



```
randSeed = 123
populationSizeN = 100
stringSizen = 16
probApplyCrossover = 1
probApplyMutation = 1.0
selectionMethod = 0
tournamentSizek = 2
fitnessFunction = 2
crossoverOperator = 0
h = False
g = False
G = False
runmode = 3
bisectionTimeout = 500
# add below to settingsfile
pauseAtBeginning = False #this just
graphing = True
w = 25 #int(stringSizen * 1)
replacementAmount=populationSizeN
```



I had trouble with Bisection Modes running extremely slowly and inconsistently, so these are the only two ones I can provide.

Summary:
Unsurprisingly interleaved trap problems are extremely hard for GAs to solve effectively. Right now I'm stuck between a rock and a hard place because without optimizations my code probably can't handle much harder problems. One thing that made this problem a little harder was the modularity needed to smoothly implement this with what I already had. My project is slowly turning into spaghetti code, which doesn't help with the efficiency problems I'm having.

I'm fairly surprised at how little diversity helped here, but I guess that speaks more to how hard this problem is.

Below is a Onemax with RTR and as you can see, Diversity actually slows down the population growth to a crawl. I know it's helpful in some situations but it seems that so much fine tuning is needed to make it work that it's hard to use.