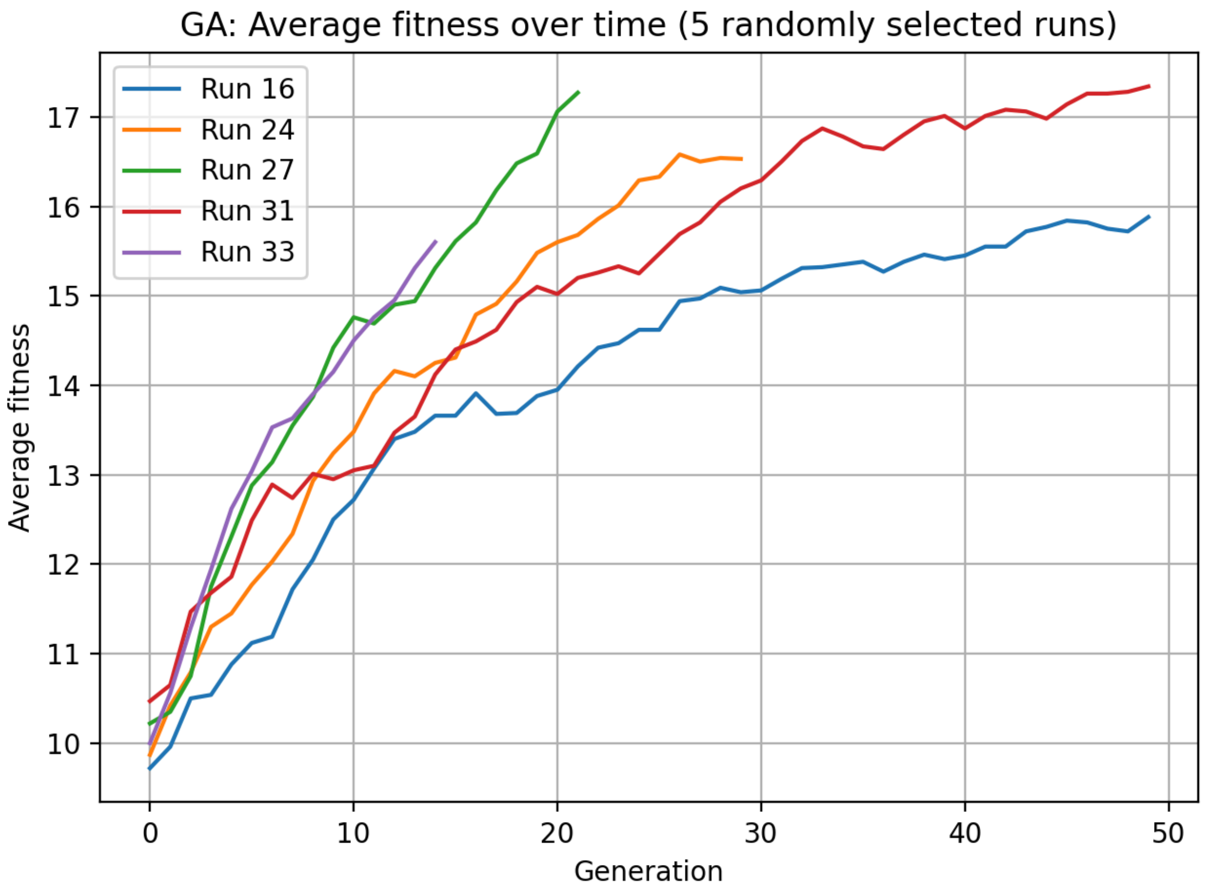
Genetic Algorithms Report - Sariah Shoemaker

Part 1 - Bit Strings

1.

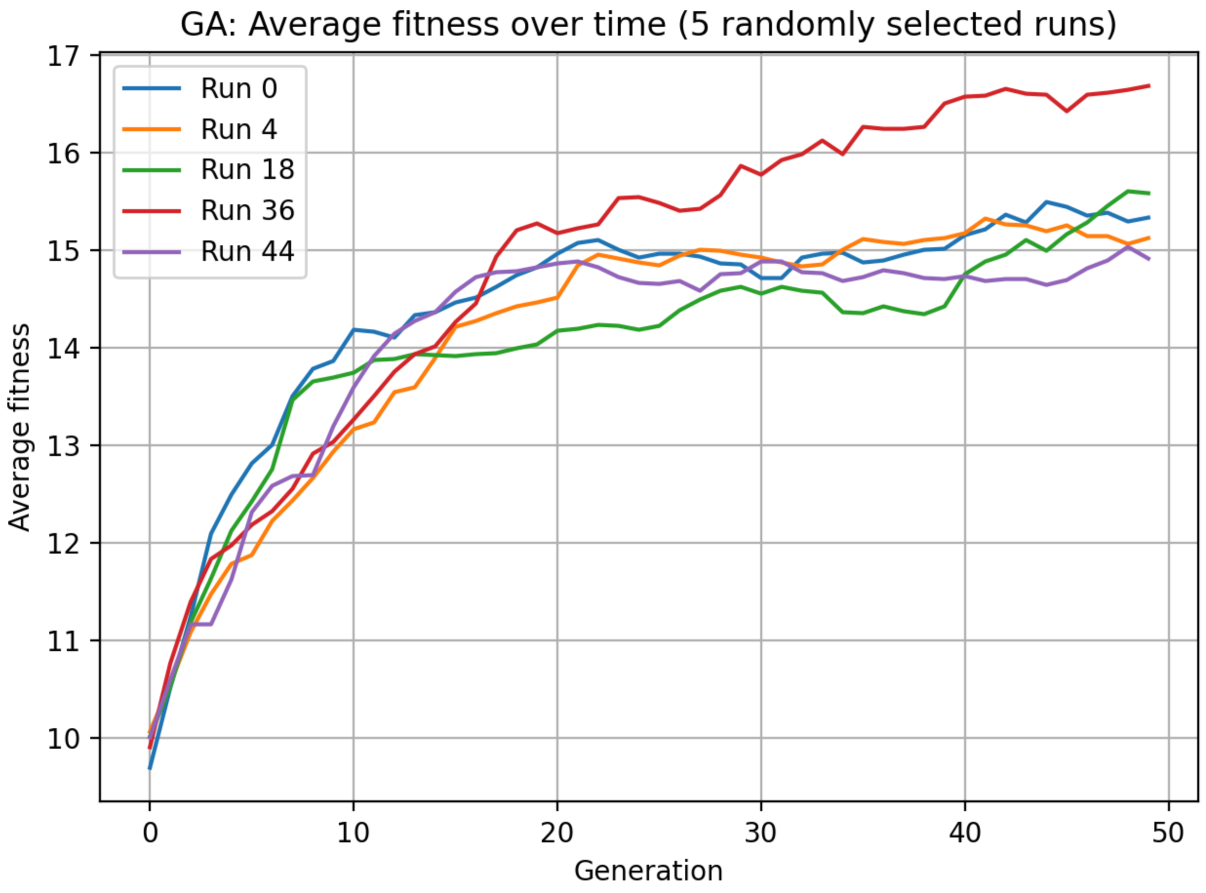


Over 50 runs (Population Size = 100, Crossover Rate = 0.7, Mutation Rate = 0.001)
Average generation found: 24.666666666666668
Min generation found: 9
Max generation found: 48
5 runs did not reach all-ones within the cap

Across all 5 runs, we get rapid improvement at the beginning. This growth slows as we get further in our generations and our genomes converge. None of the runs have an average fitness of 20, they end up around the high teens. This is because at the time we find our solution, we only have one genome with all 1s and the rest are some other combination. Run 33 and run 27 find their results rather quickly, run 31 and run 16 take almost double the number of generations.

The differences appear because of the random nature of the genetic algorithm. The population starts randomly and the mutation is random.

2.

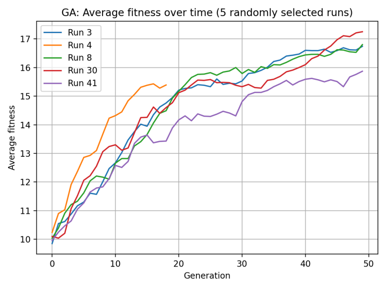
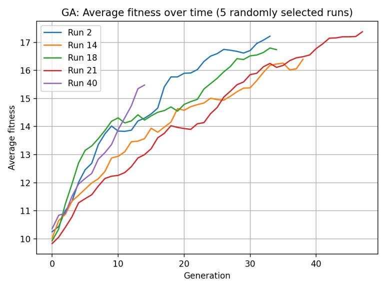
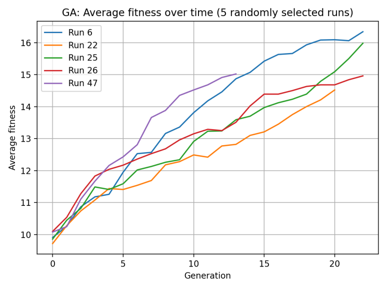


Over 50 runs (Population Size = 100, Crossover Rate = 0, Mutation Rate = 0.001)
Average generation found: None
Min generation found: None
Max generation found: None
50 runs did not reach all-ones within the cap

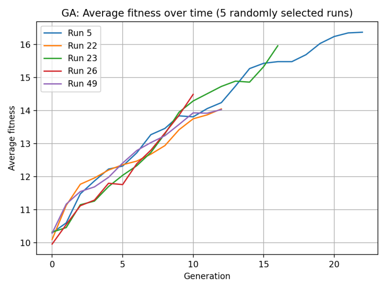
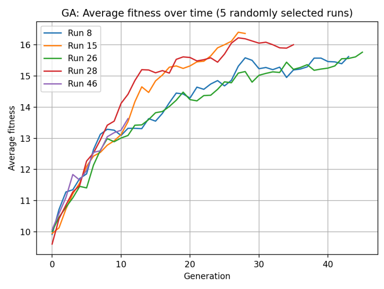
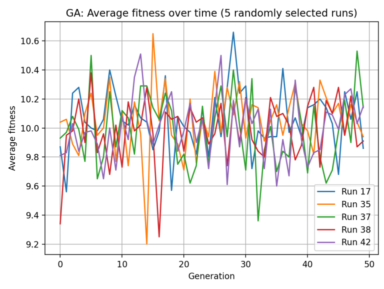
When there is no crossover, we don't get the benefits of a genetic algorithm. Instead of choosing from the best of our genomes, we are depending entirely on the randomness of mutation. Mutation alone could possibly get us to our results but it would take a lot longer. We see an increase at the beginning but as we go on in our generations we end up with peaks and valleys instead of consistent improvement.

3.

Comparison of crossover

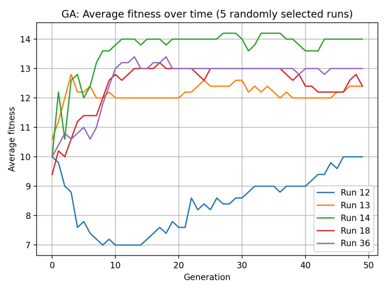
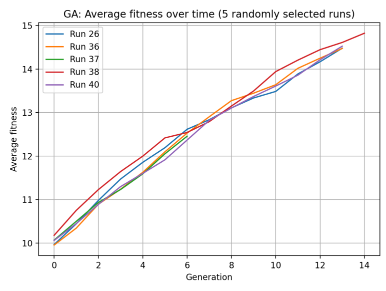
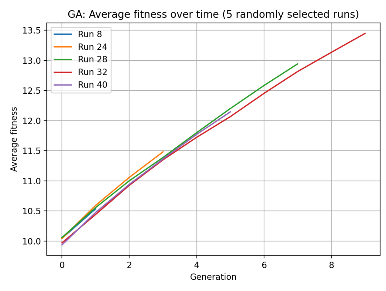
Crossover = 0.25	**Crossover = 0.5**	**Crossover = 1**
		
Over 50 runs (Population Size = 100, Crossover Rate = 0.25, Mutation Rate = 0.001) Average generation found: 31.9 Min generation found: 11 Max generation found: 46 30 runs did not reach all-ones within the cap	Over 50 runs (Population Size = 100, Crossover Rate = 0.5, Mutation Rate = 0.001) Average generation found: 29.785714285714285 Min generation found: 10 Max generation found: 49 8 runs did not reach all-ones within the cap	Over 50 runs (Population Size = 100, Crossover Rate = 1, Mutation Rate = 0.001) Average generation found: 22.2 Min generation found: 8 Max generation found: 48

Comparison of mutation

Mutation = 0	**Mutation = 0.01**	**Mutation = 0.5**
		
Over 50 runs (Population Size = 100, Crossover Rate = 0.7, Mutation Rate = 0) Average generation found: 23.270833333333332	Over 50 runs (Population Size = 100, Crossover Rate = 0.7, Mutation Rate = 0.01) Average generation found: 28.693877551020407	Over 50 runs (Population Size = 100, Crossover Rate = 0.7, Mutation Rate = 0.5) Average generation found: None

Min generation found: 6 Max generation found: 48 2 runs did not reach all-ones within the cap	Min generation found: 10 Max generation found: 47 1 runs did not reach all-ones within the cap	Min generation found: None Max generation found: None 50 runs did not reach all-ones within the cap

Comparison of population size

Population = 5	**Population = 500**	**Population = 5000**
		
Over 50 runs (Population Size = 5, Crossover Rate = 0.7, Mutation Rate = 0.001) Average generation found: None Min generation found: None Max generation found: None 50 runs did not reach all-ones within the cap	Over 50 runs (Population Size = 500, Crossover Rate = 0.7, Mutation Rate = 0.001) Average generation found: 13.6 Min generation found: 3 Max generation found: 22	Over 50 runs (Population Size = 5000, Crossover Rate = 0.7, Mutation Rate = 0.001) Average generation found: 7.18 Min generation found: 1 Max generation found: 10

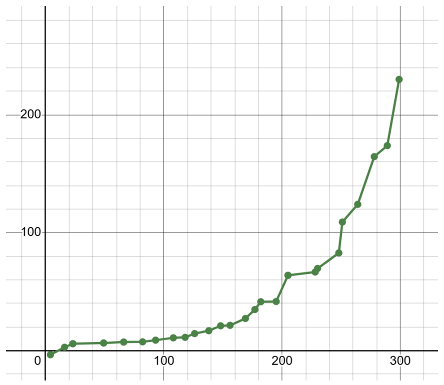
Summary

The parameter that produced the quickest solution was population size. The more genomes in my population, the less generations we can brute force our way through the problem. When the population is only 5, we don't find a solution. At 100, I averaged at the 25th generation. At 500, I averaged at the 13th generation. At 5000, I averaged at the 7th generation. The drawback to this is that it takes a lot more time to run the algorithm with more genomes.

Crossover rate also had a high impact on speed. This is because crossover picks the best candidates to reproduce. Good candidates are likely to produce good children so I want a high crossover rate. At 0, I don't get any solutions within 50 generations. At 0.25 I averaged at the 32nd generation. At 0.5 I averaged at the 29th generation. At 0.7 I averaged at the 25th generation. At 1, I averaged at the 22nd generation.

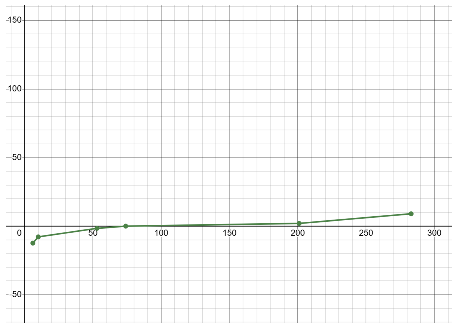
Randomness was the most interesting to me. The higher the randomness, the more peaks and valleys we end up with. This is because we could be at a really good solution that has half its bits mutated to a really bad solution or vice versa. So you want the randomness to be small. I was able to reach a solution without randomness and it averaged at the 23rd generation. I didn’t seem to get improvements even with changing the randomness a little bit, 0.001, which averaged at the 24th generation. However, I can see why randomness is still included. If I ended up in the very unlikely worst case scenario where I only had 0s, there would be no way to get to a solution without a random flipping of a bit to 1.

Part 2



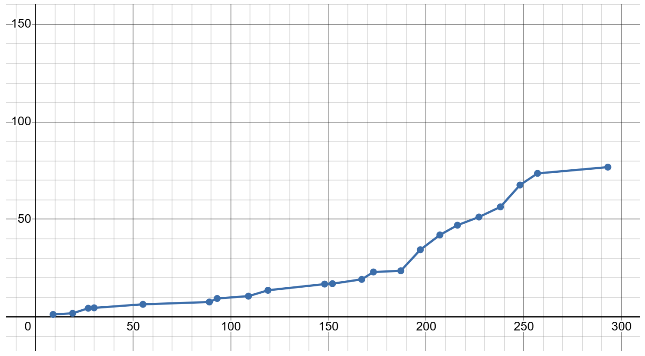
Best fitnesses from every 10 generation where population = 100, crossover = 1.0, and mutation = 0.005

This was the default experiment and it took my computer about 2 hours to run. Obviously, I have some optimization I could do to make this better. I found it interesting how further generations found much better solutions much faster. For my next experiment, I tried to do a much smaller population size with the hope it would improve my timing. It did but it came at a great cost to my best fitness.



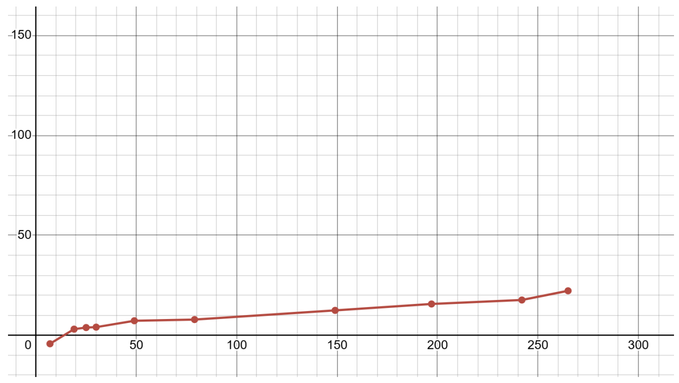
Best fitnesses from every 10 generation where population = 10, crossover = 1.0, and mutation = 0.005

Clearly, that was overkill. So I tried a population of 50 instead.



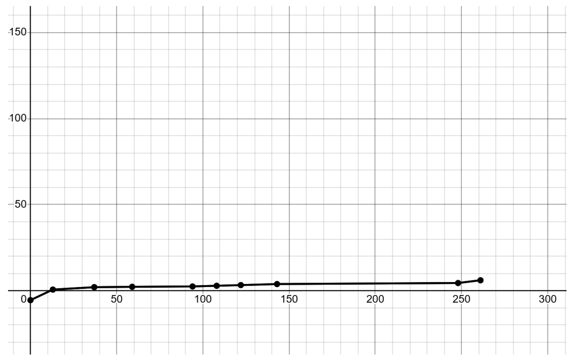
Best fitnesses from every 10 generation where population = 50, crossover = 1.0, and mutation = 0.005

It wasn't as good as the population of 100 but I was generally getting the pattern of growth that I wanted and it didn't take nearly as long to run. So I decided to give changing the crossover rate a try, I went with a crossover rate of 0.75.



Best fitnesses from every 10 generation where population = 50, crossover = 0.75, and mutation = 0.005

A lower crossover rate made things drastically worse. I decided higher was better and that matched with what I saw in part 1. The last thing I tested was making the randomness even smaller, 0.001.



Best fitnesses from every 10 generation where population = 50, crossover = 1.0, and mutation = 0.001

This surprised me, it made things almost as bad as when I made the population only 10. In the previous problem, randomness didn't seem to have a positive effect but in this problem it clearly does. I think that's because randomness is what helps Robby discover new paths.

Overall, I got the best results from the default parameters. I think I could get even better results if I went through and optimized the algorithm so it doesn't take so much time. With that optimization, I could make the population even higher and go through more solutions. If the program didn't take so long, I would also be interested in fine tuning the randomization further.