Anusha Mishra

Project 1: Genetic Algorithm

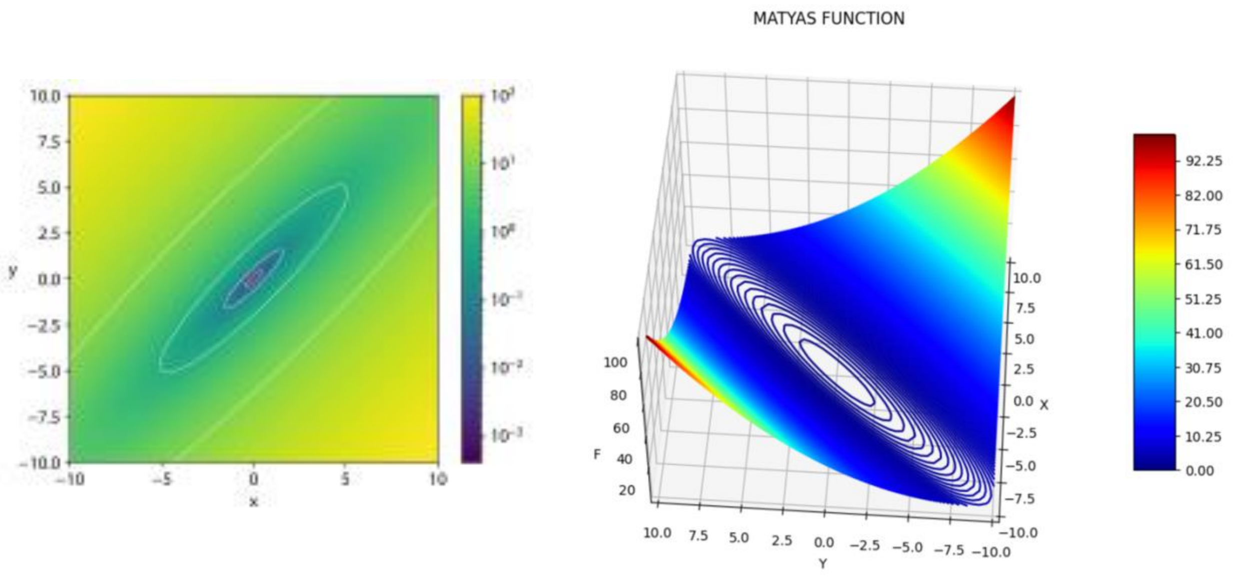
09/29/2022

Am9fq

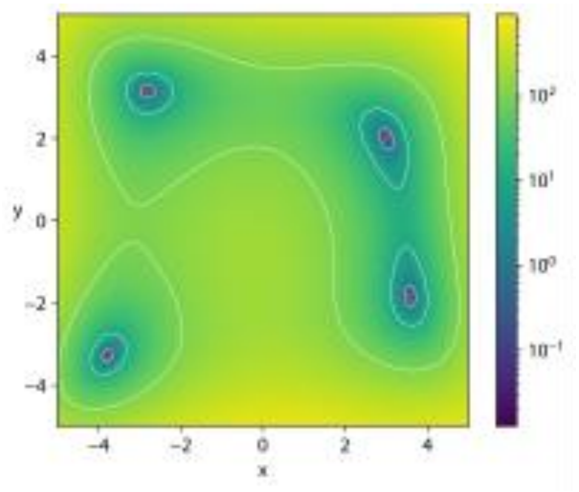
Project Description

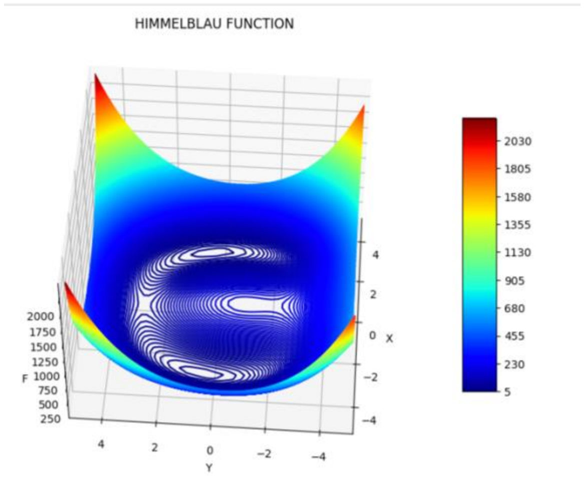
We were asked to create different genetic algorithms to test various functions used in optimization problems. Genetic algorithms follow a common template but the methods for doing each step of the template can be varied. For most problems and applications, there is no one correct combination for each. For my project, I wanted to focus of finding out how much of a difference does it make to change the various parameters that go into each strategy rather than changing the strategy completely. Would those changes be small or big? To do that, I tested various combinations of parameters. For population initialization, I chose random initialization as my strategy with the base value being 100. I varied the size of the population to be 50 individuals, 100 individuals, 250 individuals. For selection, I used roulette wheel selection. For crossover, I did 2-point crossover for the "base" GA and compared it with 3-point crossover and 5-point crossover. For mutation, I compared the different rates of mutation to be 0.05, 0.25, and 0.35. You can add elitism throughout the functions so that you only keep the most fit individuals. I compared whether using elitism throughout makes a difference compared to not using it at all. Lastly, I varied the number of generations I used to be 100 generations, 150 generations, and 200 generations.

To begin, I had to choose some functions to test my genetic algorithm on. I chose three different functions with varying levels of difficulty to optimize. The simplest function is the Matyas function. It has one global minimum at (0,0). The minimum value at this point is 0. Intuitively, it seems like it should be easy to optimize because there is only one minimum, and the range and domain of the function is not very large.

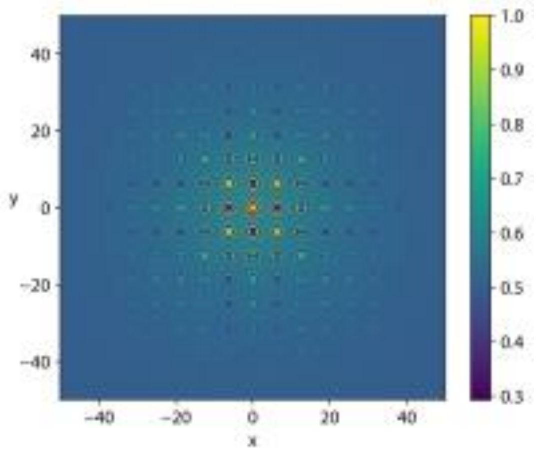


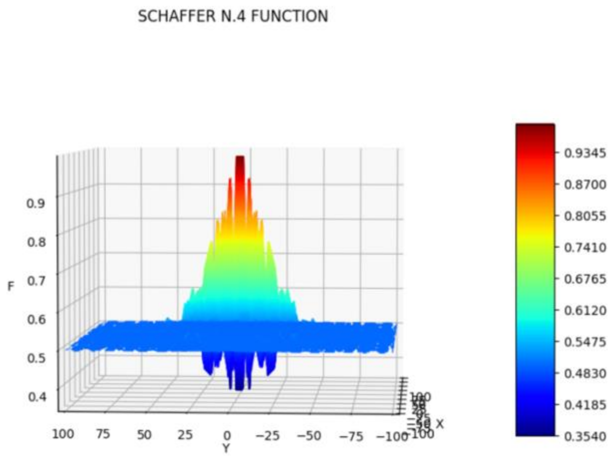
The function of medium difficulty is the Himmelblau function. It has four global minima at (3.0, 2.0), (-2.805118, 3.131312), (-3.779310, -3.283186), and (3.584428, -1.848126). They all have a minimum value of 0.0 at those points. It has a smaller range and domain from -5 to 5. This is inherently harder to optimize because there is multiple. At the beginning, I could see the algorithm either getting stuck somewhere in between one of the global minima or being scattered across the four minima.





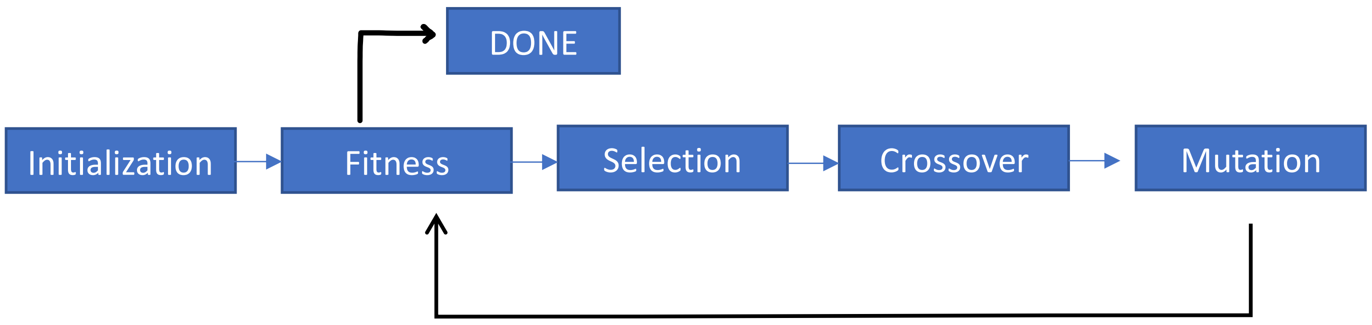
The function that is the most difficult to optimize out of the ones I chose is the Schaffer number 4 function. It also has four global minima. They occur at (0, 1.25313), (0, -1.25313), (1.25313, 0), and (-1.25313, 0). The minimum value being 0.292579 at these points. There are numerous local minima that the points may get stuck at which makes it hard to optimize.





My base algorithm has the same strategies as the algorithm Dr. Anderson gave us to study genetic algorithms. It is the same except for the elitism he put into each of his functions. This is for two reasons: I want to compare the difference elitism makes between our two implementations, and I would have made the same choices for what seems like a good combination of strategies to use for this optimization problem.

Because there are a lot of parameters to vary, I decided to try the different variations of parameters in the order of the steps that genetic algorithms are performed in depicted below.



When choosing the functions, it was important to choose functions with varying sizes of ranges and domains because having a high number of individuals may not be important for functions with small ranges. You may be able to observe convergence with less points than you would be in a function with a large domain and range. That is one of the hypotheses that I tested when varying the parameters of my genetic algorithms. For initialization, I chose random generation of chromosomes because random generation is most likely to create high diversity which should be helpful in optimization especially of the trickier functions. I initialized my population with random floats within the range of the function being used. My initial population size was 100 but I also tested 50 and 150.

In the fitness step, the fitness is determined by the output value of each random value pair put into the function. Determination of the fitness is done by computing the following formula.

$$Fitness = 1/(optimizationfunction(x, y) + 0.01)$$

Part of the fitness evaluation can involve sorting the population from least to most fit. This can be useful when you are using elitism later in your algorithm. I did not vary the way the fitness was calculated because I need to keep it the same to evaluate the differences between the variations of the algorithm I came up with.

For selection, the strategy I used was roulette wheel selection. To do this, you must calculate the cumulative fitness of all your population. You generate a random number and then you go through the fitness scores until your accumulation variable is less than the number generated. This selects a parent. You do this until you have found the desired number of parents. I chose this because again, I like the idea of using randomness to select your parent. It should give diversity and produce new solutions so that the algorithm doesn't head too much in one direction like it would if we had systematically chosen points, but the chosen points are similar and far from the global minimum. There isn't really a way to vary the parameters of this selection strategy like I can with the rest of the algorithm, so this remained the same throughout. I could have mixed in another selection type like tournament selection, but as I stated, I wanted to focus on the impact that changing the parameters can have without changing anything fundamental.

For crossover, I chose the strategy of two-point crossover to begin with for my base algorithm. Two seemed like a good number to choose because I didn't want it to have too many crossovers because that might prevent my algorithm from converging to an answer. I didn't want to have too few crossovers that I didn't encourage diversity and fitness. Later in my process, I also tested out a three-point crossover and a five-point crossover. This would allow me to see whether having a lot of crossovers can create the types of problems that I anticipated. This part of the algorithm seems that it could be associated with the number of individuals you start with.

On other words, you could tune one based on the other. I was curious to see if having a lower population size and higher number of crossovers would create similar results to having a high population size and low number of crossovers. It seems that crossovers might be a way to introduce diversity when you have a limited number of points you have but you may already have a lot of diversity from a big population size if you initialized your chromosomes like that.

For mutation, the only thing you can tweak around is the mutation rate. To perform mutations, you set the mutation rate and a number is generated at random to decide whether an individual in your population will be replaced with a new point. My base algorithm uses a 5% mutation rate. I probably would have chosen something higher, but to compare the use of elitism versus not using elitism, I kept it the same. Later, I compared a 5% mutation rate with 25% and 35% mutation rate.

Lastly, you also have control over the conditions with which you want to end your algorithm. You could end your algorithm when it converges. I chose to stick with the theme of picking the number of generations that the algorithm would go through. The algorithm restarts at the fitness evaluation stage after every generation until the desired number of generations has been reached. Dr. Anderson used 100 generations for his genetic algorithm. I compared this with 150 and 200 generations. The reason I didn't test a smaller number of generations it is because I wanted to give the algorithm adequate time to converge to an answer or the closest that it was going to get.

Implementation

Himmelblau Function

1. Elitism vs No Elitism
2. n=50 versus n= 100 vs n= 150 population
3. 2-point vs 3-point vs 5-point crossover
4. 5% vs 25% vs 35% mutation rate
5. 100 vs 150 vs 200 generations

Matyas Function

1. n=50 versus n= 100 vs n= 150 population
2. 2-point vs 3-point vs 5-point crossover
3. 5% vs 25% vs 35% mutation rate
4. 100 vs 150 vs 200 generations

Schaffer #4 Function

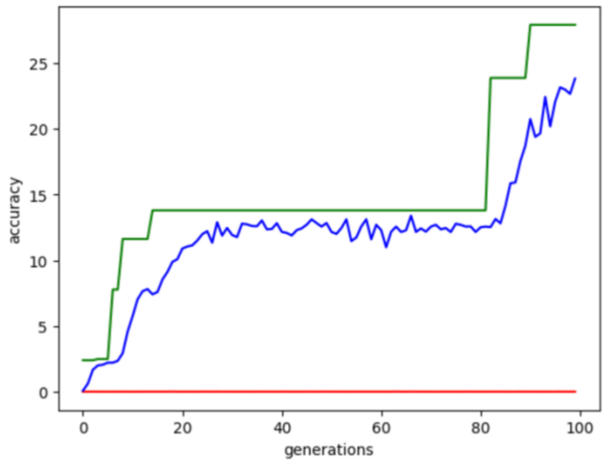
1. n=50 versus n= 100 vs n= 150 population
2. 2-point vs 3-point vs 5-point crossover
3. 5% vs 25% vs 35% mutation rate
4. 100 vs 150 vs 200 generations

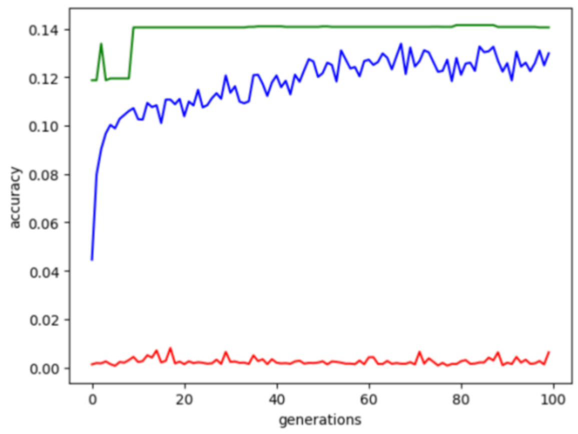
I did my experiments in a tournament style. First, I tested the comparison between elitism and no elitism on the Himmelblau function. From there, I tested each varied parameter in a three-way tournament. The better performing of the first two parameters is tested with the last parameter. This best out of the three is kept for the following tests of the function. For the parameters that haven't been tested yet, they are kept at their base settings until they are tested.

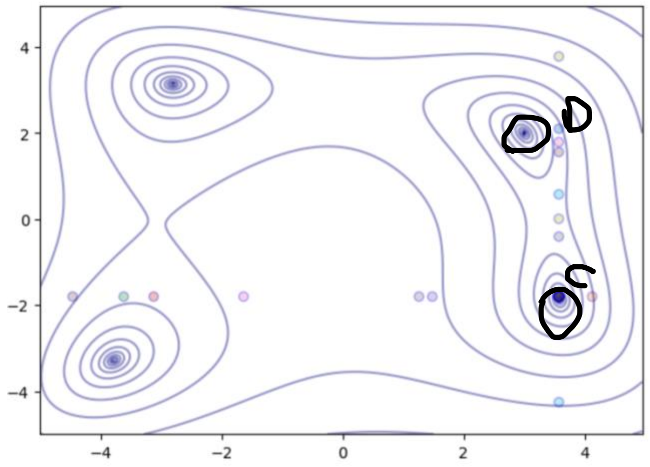
Experiments

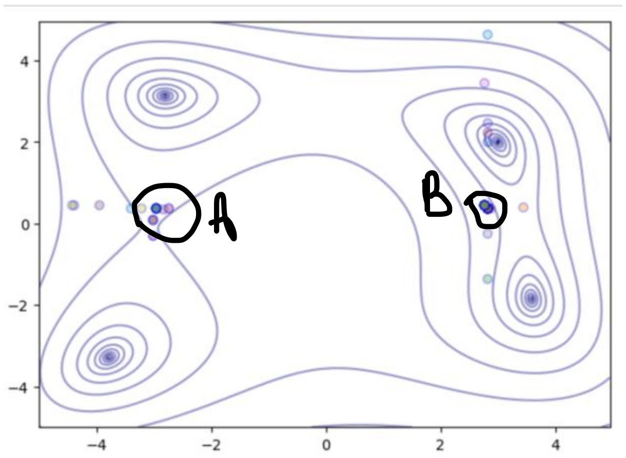
Elitism

No Elitism



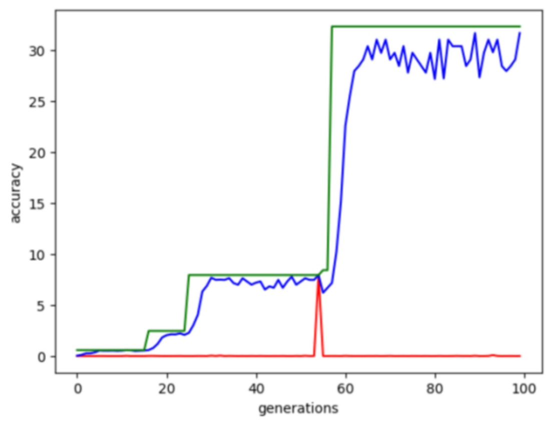




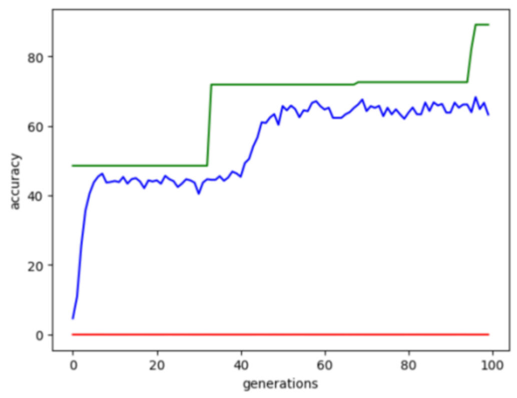


The right column shows my trials. The points are mostly going to points A and B. Mine has a low accuracy. In the trial with elitism, the points are converging to one of the minimums at point C. The improvements on accuracy level out on the no elitism side quickly. Given this result, the rest of the Himmelblau trials have Elitism.

Elitism, Population= 50

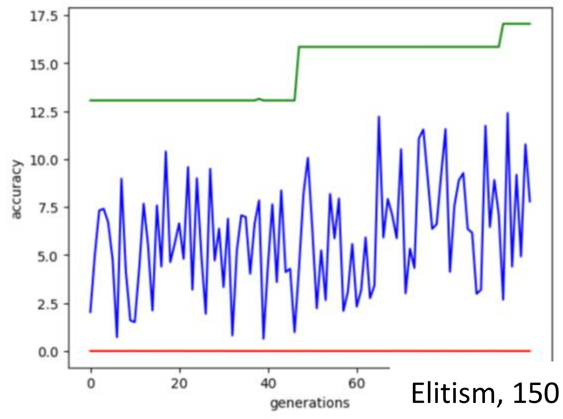


Elitism, Population= 150

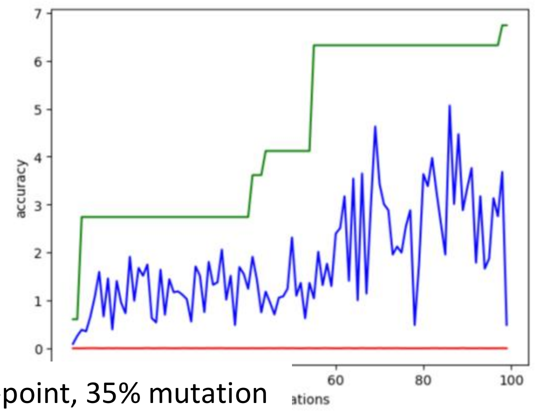


The graphs of the points are not included but they converged to the point D shown in the graphs on the previous page in both population sizes. The population size of 150 is more accurate so it's chosen for the future trials.

Elitism, Population= 150, 3-point crossover

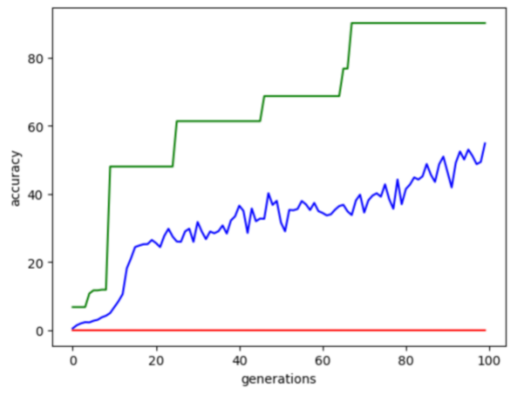


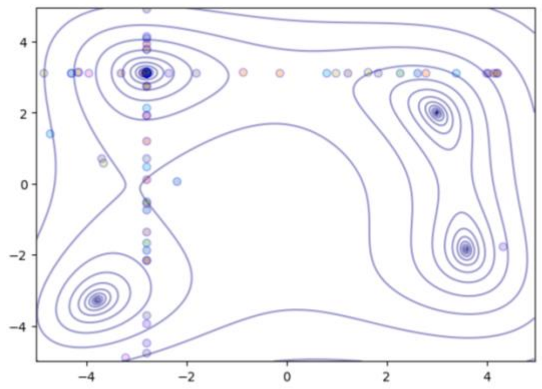
Elitism, Population=150, 5-point crossover



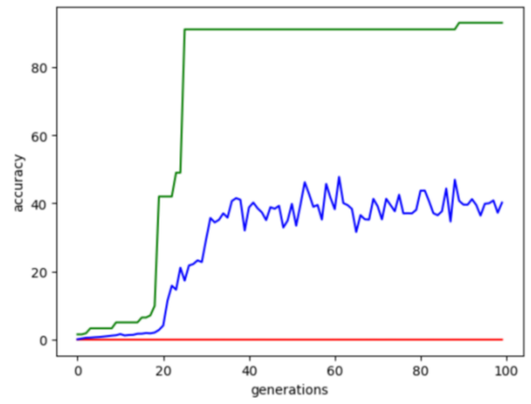
Elitism, 150 pop., 2-point, 35% mutation

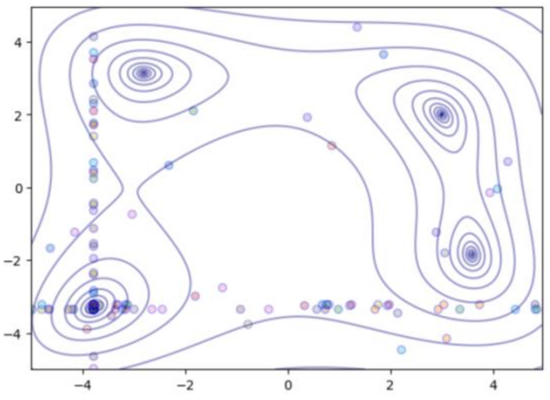
The frequent spikes show that the algorithm gets lucky occasionally, but that luck disappears quickly because of the frequent crossover. 2-point crossover seems to be the ideal





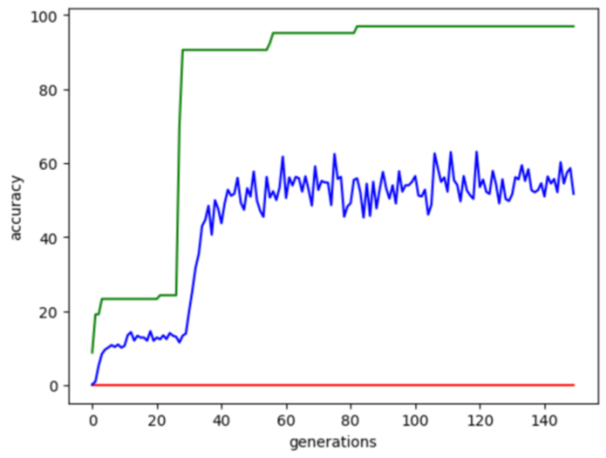
Elitism, 150 pop., 2-point, 25% mutation number of crossovers for this function. The graph is not included but you can see that the algorithm is having trouble converging to any one point.

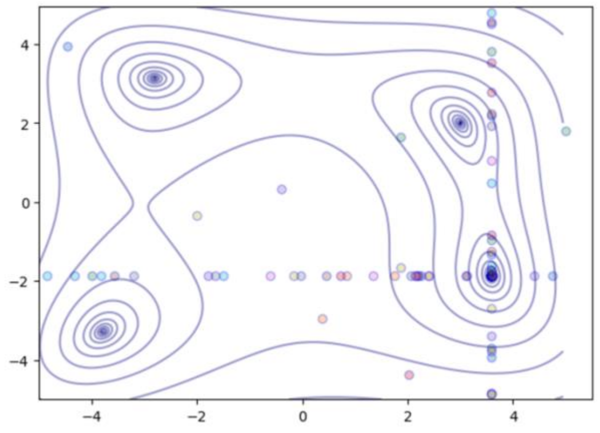




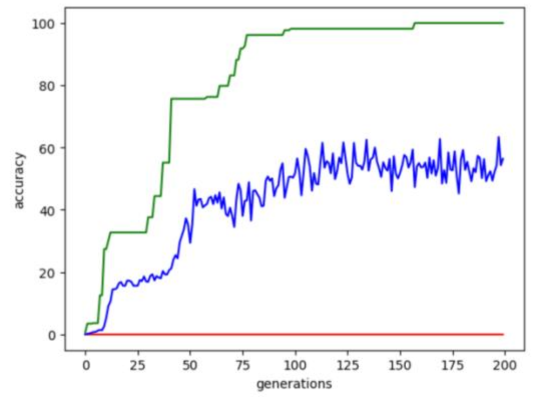
The average accuracy is similar between both mutation rates, but it seems like they converge to different minima. The average fitness is staying level with a 35% mutation rate but the maximum accuracy in each generation is much higher she got that than the average. A 25% mutation produces a steadily improving accuracy rate. That is the reason that 25% is chosen as the rate going forward.

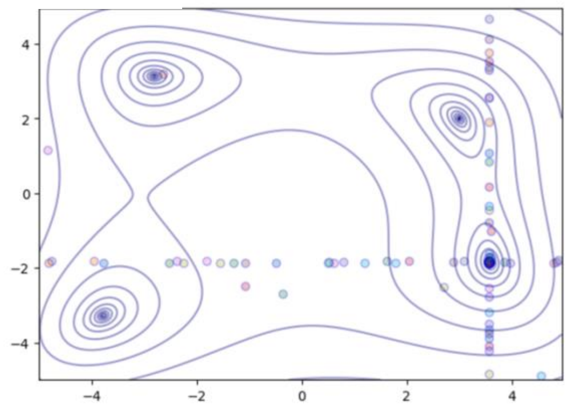
Elitism, 150 pop., 2-point, 25% mutation, 150 generations





Elitism, 150 pop., 2-point, 25% mutation, 200 generations

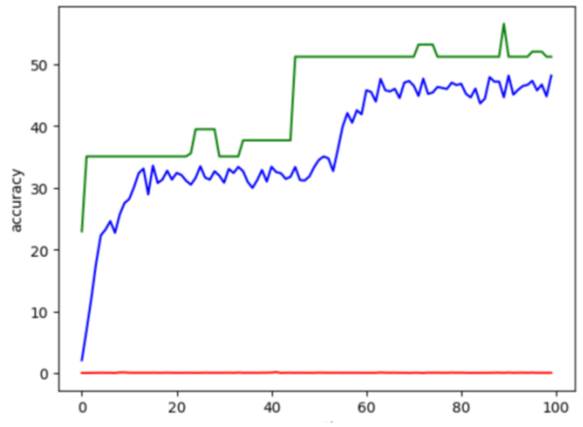


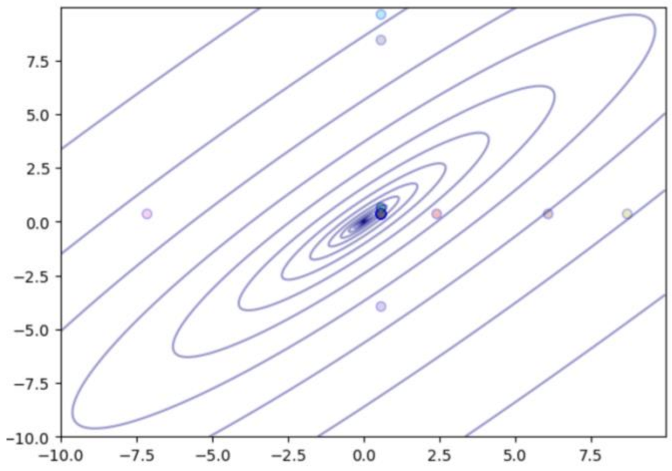


At this point we see that the maximum accuracy line is approaching 100 in both number of generations. This means that the randomness allowed for my algorithm to have some points that were very close to a minimum. Although we saw convergence before, our algorithm is getting even closer to the exact values. The average accuracy remains steady in both variations of the generation parameter. At this point either 150 or 200 generations produce near equal results. I picked 150 as the final chosen value because you don't need to do the 50 extra generations to get the same result more computational power would create. The best combination of parameters to optimize the Himmelblau function according to my trials and tested parameter values is a genetic algorithm that uses elitism, a population size of 150, a two-point crossover method, a 25% mutation rate, and 150 generations.

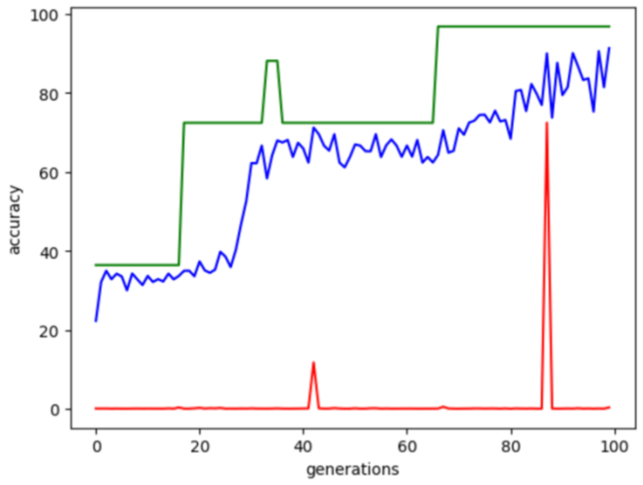
Now we can examine the Matyas function in a similar way, however, I did not test Elitism in the last two functions. In reality, I tested the Matyas and Schaffer functions first, otherwise I would have factored Elitism into all the tests. I did the tests starting from the base parameter values like I did with the Himmelblau function.

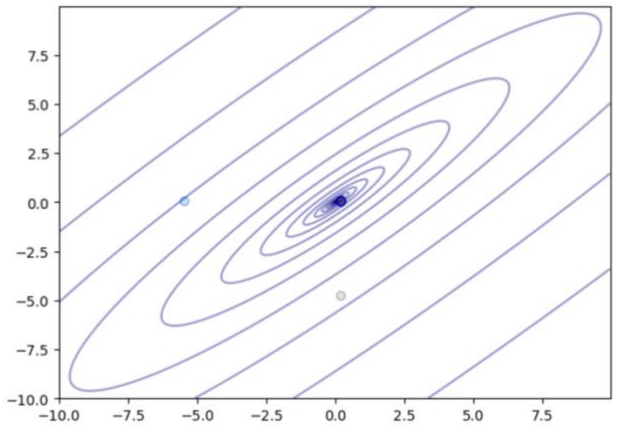
No Elitism, Population= 100



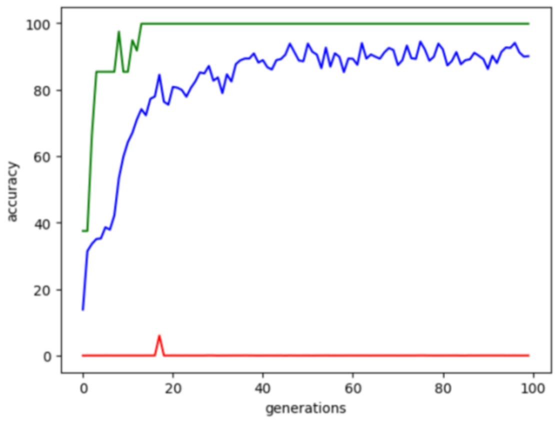


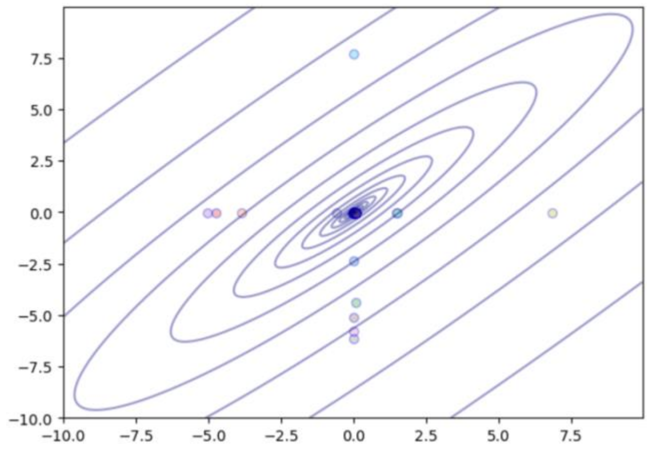
No Elitism, Population= 50





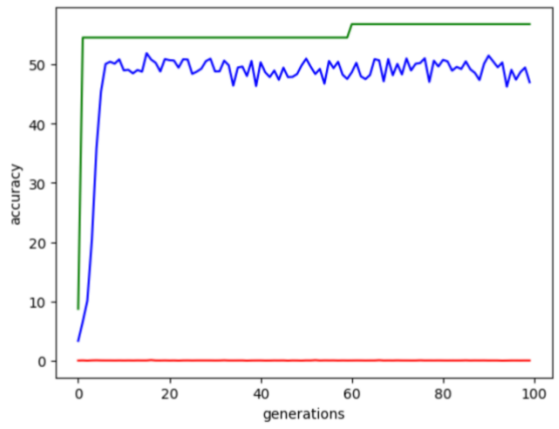
No Elitism, Population= 150



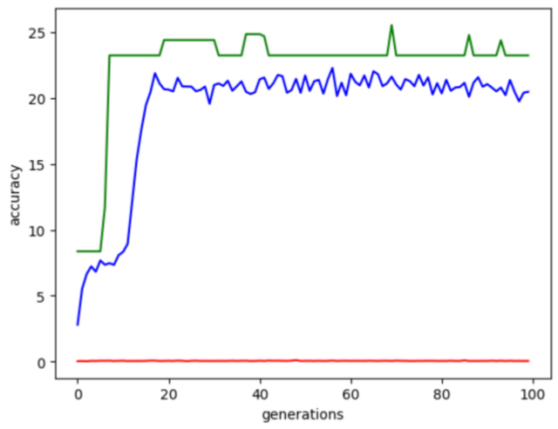


All of population sizes were able to converge very close if not fairly close to the minimum. It seems like you don't need lot of manpower when there is only one minimum. Nevertheless, the best value seems to be population 150. The higher number of points seems to lead to more precision in finding the minimum.

No Elitism, Population= 150, 3-point crossover

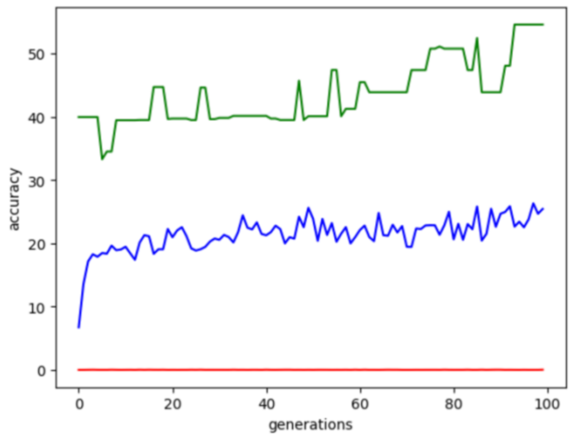


No Elitism, Population= 150, 5-point crossover

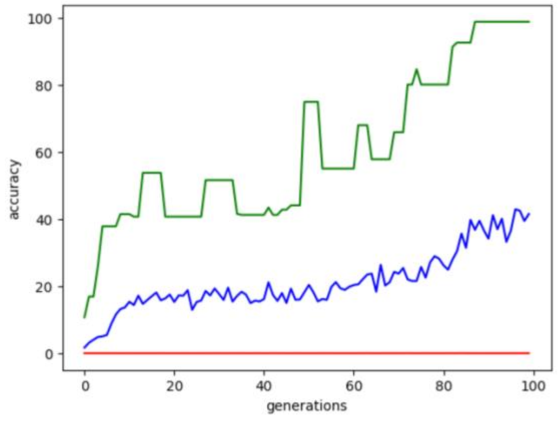


Increasing the number of crossovers done significantly reduces the accuracy. The graph containing the points is not included, but the inaccuracy is reflected there as well. Yet again, it is best to keep the number of crossovers to 2.

No Elitism, Pop. = 150, 2-point crossover, 25% mutation

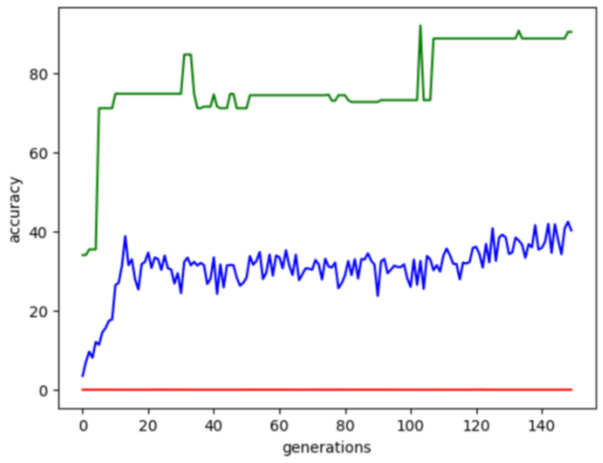


No Elitism, Pop. = 150, 2-point crossover, 35% mutation

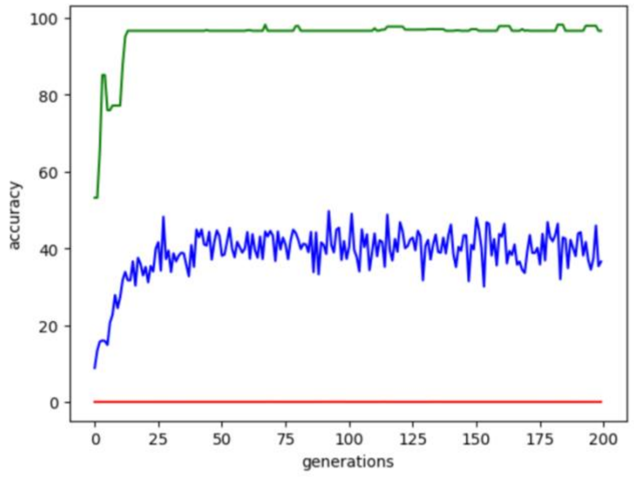


In this case having the higher mutation rate seems beneficial. Although the average accuracy line is not increasing by munch it still has a positive slope. The maximum accuracy is also approaching 100. However, that means less than the average accuracy approaching 100.

No Elitism, Pop. = 150, 2-point crossover, 35% mutation, 150 gen.



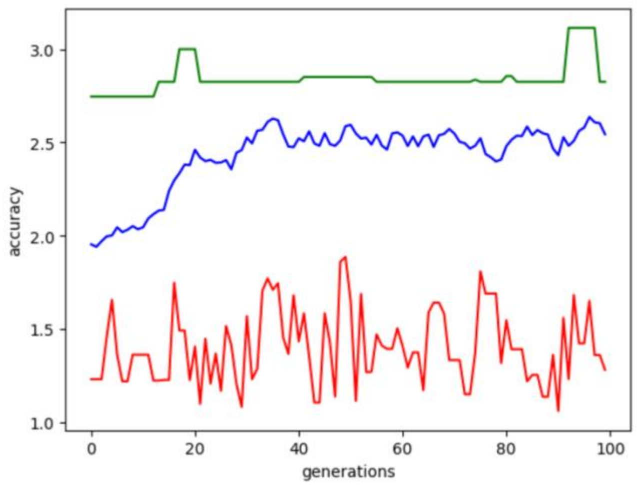
No Elitism, Pop. = 150, 2-point crossover, 35% mutation, 200 gen.



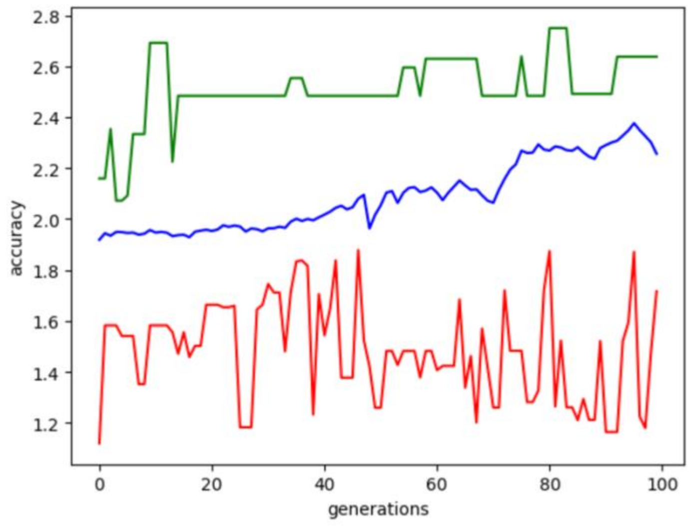
Doing 200 generations instead of 150 doesn't make much of an improvement on the average accuracy. In the same vein there is not much of a difference between 100 generations and 150 generations. The gap between the maximum accuracy and the average accuracy remains the same in both cases. The slope of the average accuracy in 150 generations levels out, but in the last 50 generations in the accuracy graph for 150 generations the slope is slightly positive. for that small improvement I chose 150 generations as the final value. For the Matyas function, the ideal combination without factoring elitism is a population size of 150, a two-point crossover, 35% mutation rate, and 150 generations.

Finally, I tested the Schaffer function, the hardest of the three optimize, in a similar fashion to the Matyas function starting with the base parameter values.

No Elitism, Population= 150

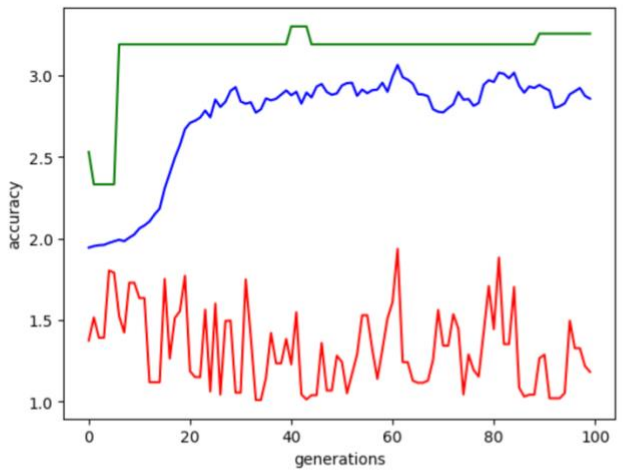


No Elitism, Population= 200

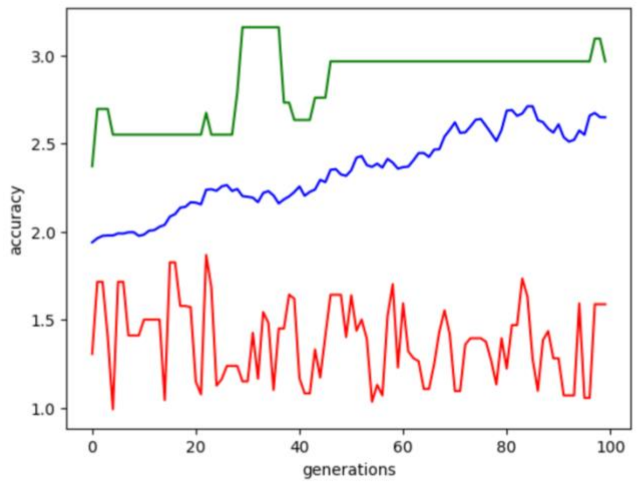


Immediately, we can see that our algorithm has much less accuracy on this function. A population with 50 individuals is not included because the accuracy is very low. Given this result, 200 individuals are chosen because of the slightly positive slope that the graph for a population size of 200 has on the average accuracy line.

No Elitism, Population= 200, 3-point crossover

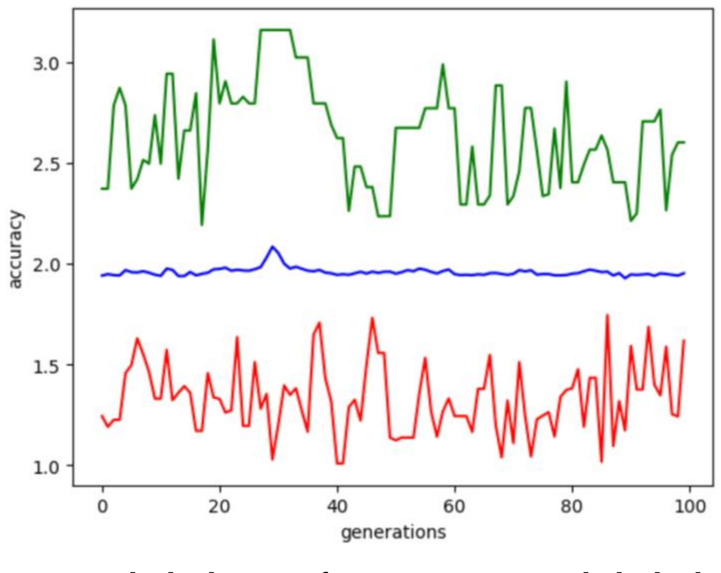


No Elitism, Population= 200, 5-point crossover

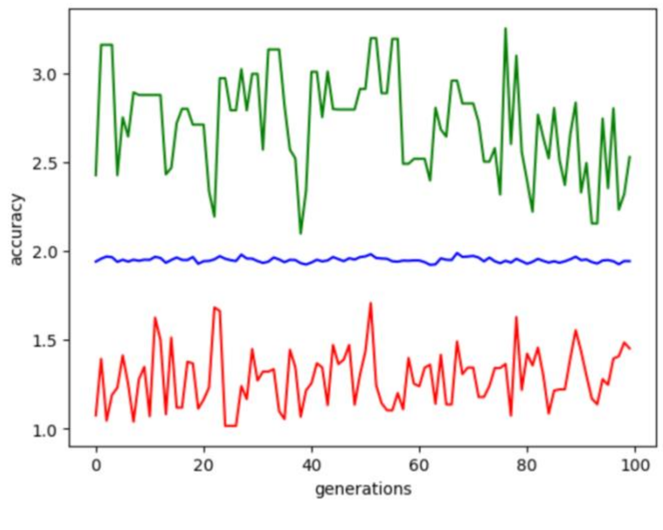


If we look at the average accuracy line in the five-point crossover, we see that having a high number of crossovers is helpful. The average accuracy line is improving steadily unlike the three-point crossover and add a better rate than the two-point crossover. We can use the five-point crossover going forward.

No Elitism, Population=200, 5-point crossover, 25% mutation

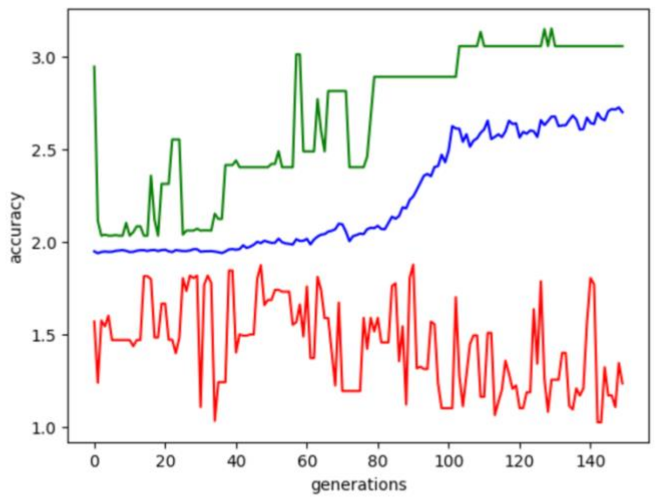


No Elitism, Population=200, 5-point crossover, 35% mutation

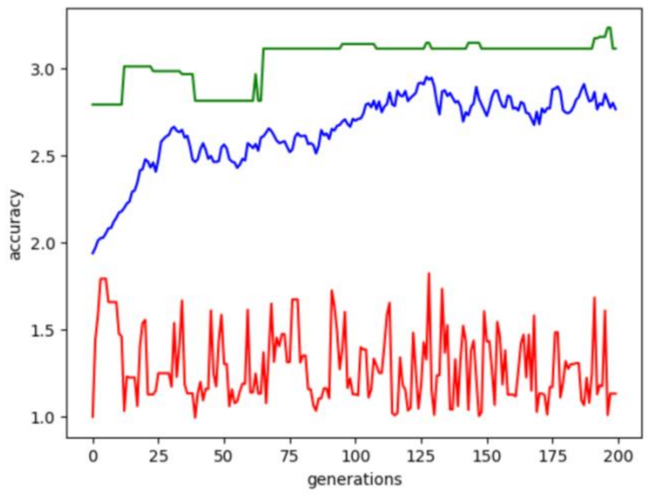


The high rates of mutation mixes with the high number of crossovers and completely throws off the algorithm. In this case, it is best to keep the mutation rate low and keep it at 5% like the baseline value.

No Elitism, Pop.=200, 5-point crossover, 5% mutation, 150 gen.



No Elitism, Pop.=200, 5-point crossover, 5% mutation, 200 gen.



Number of generations doesn't seem to help much except a little bit in the case of using 150 generations. Even after all that tweaking it doesn't seem like we can change the accuracy of our algorithm much by tweaking the parameters in this case because this function is so difficult to deal with. The ideal set of parameters for this function is a population size of 150, a five-point crossover, A 5% mutation rate, and 150 generations.

Conclusion

At the beginning of this project, one of my hypotheses was that having a high number of points may not be important for functions with small ranges. While that seems to hold because the function with the largest range and domain, Schaffer function, had a larger population value that works the best for it. after going through the experiment, I can tell that it isn't exactly about the range itself. The Schaffer function does better with more points because there are so many local minima on it. It needs a higher diversity within its points so that it doesn't get stuck on the local minima. Even with my largest value for the population value, the accuracy is very low. The little accuracy my algorithm has seems to be associated with luck as we can see from the random spikes in the minimum accuracy, average accuracy, and maximum accuracy. In the future because my population sizes were not that buried off, knowing what I know, I would test a wider range of population numbers especially for complex functions like the Schaffer function.

Another thing that got confirmed with my experience is that having a high number of crossovers is not helpful when you don't need a lot of diversity. It was helpful to the Schaffer function for the reasons explained above but in the other cases it was harmful to the accuracy of the algorithm. from my trials, it seems that two-point crossover are a good starting point for most problems. If you see a need to tweak it later, you could do that as you see fit.

My intuition was correct that a higher rate of mutation may be helpful to the algorithm. In the outlier case of the Schaffer function, where there was already a high number of crossovers being used and a high mutation rate along with it would mess up the function. Given that, if you already have a lot of diversity, it is not necessary to have a high mutation rate. I was right that it prevents the algorithm from converging. The other two algorithms did better with 25% and 35% mutation then the baseline of 5% mutation.

as we were taught in class, there is no one specific way to implement a genetic algorithm that works for every case. In cases where there are a lot of local minima and other reasons for difficulty and optimization, there will have to be a lot of combinations done. Otherwise, you don't have to put in as much manpower to find the right combination. Certain key choices can make the difference.