PROJECT 1:
GENETIC ALGORITHMS

ANDY VARNER
12153597
FALL 2022

INTRODUCTION / PROJECT DESCRIPTION

The purpose of this project is to gain a meaningful introductory understanding of the topic of Genetic Algorithms (GA). The background I bring to this work is in electromagnetics and material design. There is increasing interest in this field in using artificial intelligence for the design of optical metamaterials, and I would like to understand and contribute to the current body of work. Although I have some programming experience, I am effectively a beginner in the field of computational intelligence. Thus, I would like to learn how genetic algorithms function on a basic level so that I might begin to understand what kinds of questions in the field of optical metamaterial design may be answered by a genetic algorithm. I will achieve my goal if I can demonstrate a command of the theoretical foundations of the field, the ability to implement and tune the algorithm, and the ability validate its performance. Beyond this, I hope to gain an intuition for how the tuning of certain parameters impacts a particular problem. I'd like to be able to answer the question, "Is this a good problem for a genetic algorithm? Why or why not?" I will challenge myself and solidify this intuition through the implementation of an Island GA.

I begin by outlining the basic genetic algorithm in the Background: The Basic Genetic Algorithm section. I go into some detail about the various implementation choices I make in terms of operators in the Implementation section. I also describe Island GAs and how we use them in the Implementation section of this report. I describe my experiments and their results in the Experiments and Results section. Experiments and Results begins with a trivial but instructive problem: I. Minimize X^2. Once this is established, I extend the problem to two variables and investigate how the algorithm solves some well-understood test functions for single-objective optimization: II. Minimization of the Matyas Function, III Minimization of Himmelblau's function, and III. Minimization of The ackley function . For the graduate student task, I investigate the utility of island GAs with some emphasis on Himmelblau's function which will be discussed later. This gives me the opportunity to implement an island GA in addition to a standard GA and to make a qualitative comparison. I aim to answer why and if an island GA is superior for this particular set of problems, and if not, to consider what sort of problem may be best-suited to the island GA. I conclude with a Summary of Results and Conclusion section followed by Future Work.

BACKGROUND: THE BASIC GENETIC ALGORITHM

The GA falls under the umbrella of Evolutionary Algorithms (EA) in the field of Computational Intelligence. EAs, inspired by nature, attempt to solve an optimization problem by developing and implementing a mathematical approximation to evolution. The genetic algorithm, then, seeks to model genetic evolution by expressing the characteristics of individuals as genotypes [2]. Genetic algorithms can be thought of as random search with heuristics added in; that is, we might try to optimize a function by exploring our domain space, not entirely randomly, but by employing mathematical approximations to the techniques observed in genetic evolution, namely, the encoding of chromosomes as a means of information exchange. For GAs, those techniques are identified as selection, crossover, and mutation.

The algorithm is performed on an encoded set of values, possible solutions to the problem, which exist within some determined search space. These possible solutions are thought of as chromosomes, which contain a set of n genes; for example, a two-dimensional problem has chromosomes which contain n=2 genes, let's say an x gene and a y gene. The fitness function, f(x,y), returns a value which we call the fitness of that chromosome. For our purposes, each chromosome has n number of genes and one fitness. The fitness is determined for each chromosome of each generation and used to determine reproduction. Let's consider this in light of the genetic algorithm, which is generalized as follows:

- Let the number of generations, or epochs, be 0.
- Initialize the population, C(0).
- Do until done:
  - *Evaluate the fitness, $f(x_i)$, of each individual, x_i
  - Perform Selection
    - Crossover
    - Elitism
    - Mutation
  - Select a new population, C(++1).
  - Advance to the next generation.

We go into some detail about the operators of this algorithm, as well as the Island GA, in the next section, Implementation.

*For a two-dimensional problem, $f(x_i,y_i)$, of each individual consisting of genes x_i, y_i

IMPLEMENTATION

Now that we have established the foundation of the basic genetic algorithm, we will go into some detail about specific operators chosen for this project, and finally, provide a framework for the island genetic algorithm.

POPULATION INITIALIZATION

The initial population is chosen randomly, and the size of the population and number of generations are chosen such that there is enough "room" for the algorithm to exploit and explore within the search space. Choosing the initial population randomly within the parameters of the search space is important, because it ensures we start with a uniform representation of all possible solutions. For this project, each function that we test our algorithm on is defined on some domain, so that domain will be the search space from within which we initialize our population.

FITNESS

The fitness, $f(x_i, y_i)$ (or $f(x_i)$ for one dimension), evaluates to some point on the function. The optimization problem we encounter in this project is always a minimization problem, so a fitness approaching the minimum is a fitness we might think of as describing the possible solution. In short, the more "fit" a chromosome, the closer it is to the solution, or a solution. Depending on the number of minima, the algorithm could always "think" it has found the global minimum, when really it is trapped in a local minimum. Consider the red dot in Figure 1, which has converged on a local minimum and needs a strategy to "escape" and discover the true global minimum. To mitigate this problem, a major goal of the GA is to seek diversity by exploring through sufficient diversity of chromosomes. We can find some confidence that our GA does this by plotting fitness vs. generations and observing a large distance between the minimum and maximum fitnesses for the life of the algorithm, which indicates large diversity.

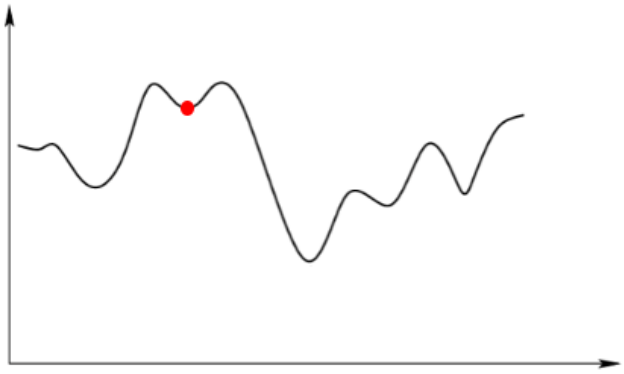

FIGURE 1. GRAPH EXAMPLE WITH MULTIPLE MINIMA

SELECTION

The purpose of selection is to produce, from the current population, a next iteration, or generation, of possible solutions to the problem with the goal that the chromosomes inherit genes with the best fitnesses. Selection can be thought of as reproduction in the natural-world analog. In the algorithm, selection considers the fitness of each chromosome and makes a decision about which parents will reproduce.

There are many well-defined selection operators available to us: Random selection chooses randomly, with no regard to fitness. Tournament and Rank-based selection are operators that do make use of fitness, but we will not consider for this project. We do make use of Proportional selection, also known as Roulette wheel, a strategy by which the probability of an individual being selected for reproduction is proportional to its fitness, according to the following distribution [2]:

$$\text{Prob}(\overrightarrow{x_n}) = \frac{F_i(\overrightarrow{x_n})}{\Sigma_{n=1}^{N} F_i(\overrightarrow{x_n})}$$

Where $\text{Prob}(\overrightarrow{x_n})$ is the probability that some individual $\vec{x}_n$ will be selected. $F_i(\overrightarrow{x_n})$ represents the fitness of each individual. So the probability that any individual will be selected is proportional to the ratio of its fitness to the average fitness of the population. Furthermore, the fitness values are normalized such that the probability distribution can be thought of as a roulette wheel or a pie, where the larger the selection probability of an individual, the larger the slice of the pie. We then "roll" the wheel by the following algorithm:

- Let the chromosome index = 1.
- Let sum = $\text{Prob}(\overrightarrow{x_n})$.
- Choose a uniform random number, $\zeta \sim U(0,1)$.
- While sum < ζ
  - Advance chromosome index
  - Advance sum
- Return the selected individual, $\overrightarrow{x_n}$

An important selection operator to consider, which we explore in this study, is elitism. Elitism ensures that the most fit parents survive to the next generation. We will evaluate the algorithm both with and without elitism in order to understand its utility and performance. We can choose the percentage of most-fit parents to survive to the next generation; when we implement elitism, we begin by preserving 20% of the most elite.

CROSSOVER

Crossover is the stage at which the selected genetic material is combined to create the new generation. We discussed 1-point, 2-point, and k-point crossover strategies, but the strategy we examine in this work is called simplex crossover, which is a kind of averaging strategy. For each parent chromosome, we randomly select genes for reproduction. Reproduction is then done by subtracting the mean value of each gene from the current gene, multiplying that value by a random uniform number, and adding the product to the initial genetic value. The next generation is assembled from genes that have been through this averaging transformation. To clarify this,

consider the arbitrary chromosomes, $x_1 \ldots x_4$, in an arbitrary vector space in Figure 2. The mean, x', is calculated by the following:

$$x' = \frac{1}{n_\mu} \Sigma_l^{n_\mu} x_l$$

where n_μ represents the fitness of the individual. The vectors $\mathbf{v_1 \ldots v_4}$ represent the Euclidean distance between the mean and the corresponding individual and are calculated by:

$$\boldsymbol{v_i} = x_i - x'$$

We determine the child, $\tilde{x}$ by:

$$\tilde{x} = x' + \Sigma_{l=1}^{n_\mu} w_l$$

where w_l represents a uniform random number between 0 and 1.

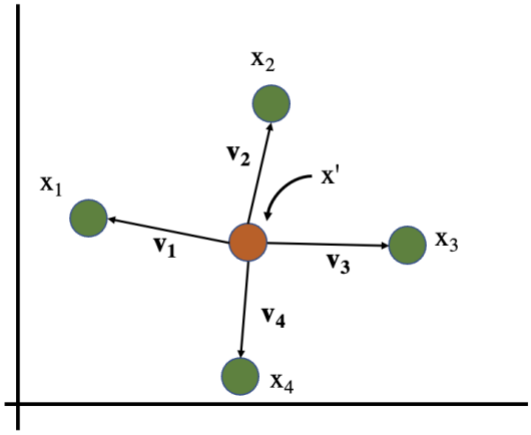

FIGURE 2. SIMPLEX OPERATOR IN AN ARBITRARY SEARCH SPACE

MUTATION

While there is some randomness built into the algorithm (e.g. we initialize with a random population, and we use a uniform random number to complete simplex crossover), truly powerful exploration that comes from randomness is built into the algorithm with mutation. This operator randomly changes the values of genes in a chromosome, generally, by the addition of a Gaussian random value, $N(0, \sigma_{nx}^2)$ to the gene [2]. This introduces entirely new genetic material into the population that is not inherited, it just materializes (because we program it to materialize). In this study we explore mutation in two ways: We define a *mutation scale* – this is the degree to which we are changing genes, so for any particular gene, a high mutation scale will change the value by a lot, and a low mutation scale will perturb the value by a small factor. We also define a *mutation rate*, which is the frequency with which mutation occurs. In this study we explore changing these values to see how our algorithm performs with varying degrees, and rates, of randomness.

ISLAND GENETIC ALGORITHMS

Island genetic algorithms offer a topology that allows multiple GAs to run in parallel, such that operations may be conducted differently for each individual GA, and the GAs communicate with each other via migration so that genetic material may be shared among the populations. A diagram illustrating the specific topology to be implemented in this project is

shown in Figure 3. Because each island is connected, this is an example of the star topology. For the designer of an island GA, the already choice-rich territory of the basic genetic algorithm becomes even more saturated with options. The number of islands represents the number of independent GAs which can be designed to optimize the most desirable characteristics of any given operator. The exploration of these choices is an ambitious next step to gaining one's footing in GAs, but it will provide the experience needed to increase intuition concerning the fine details of how these algorithms work, and what kinds of questions they might be able to approach with a certain cleverness.

As we implement the island GA, we would like to know, what advantages do we get by delegating certain operations to certain islands? How can an island GA uniquely solve a problem, and is there some problem for which it is superior to a standard GA? How do we quantify the success of an island GA? Because there are so many factors to consider, we simplify the problem such that we primarily explore mutation. Thus, two islands are defined by whether they use mutation, with the third using moderate mutation. This acts almost like a control, since as the GA we explored for the first part of this project, it is the best understood. Setting up the islands in this manner, we hope to hone our understanding of exploiting – little mutation – vs. exploring – heavy mutation.

To build this, we need to tweak our algorithm a bit. First, we determine how to initialize the subpopulations. If we wanted to delegate each island to explore a subset of the search space, we could give each island its own separate chunk of the domain (e.g. for a search space of [0,11], Island 1 would get [0,3], Island 2 would get [4,7] and Island 3 would get [8,11]). This is an interesting strategy for a problem with a nuanced search space. We choose to initialize the population randomly since we are the most interested in looking at the effect of mutation on our islands independent of other factors.

The other major consideration is how do we implement migration? This is a two-part question: How do we choose individuals to migrate, and how do we choose which islands they migrate to and from? Both could also be chosen randomly— we could choose individuals at random to migrate to a random island. Alternatively, we could borrow some strategies from selection. In this project, we define the fitness of each individual island by considering the median fitness of its population for any given generation. The "best" island has the highest median fitness and the "worst" island has the smallest median fitness. So the islands are chosen according to their fitness. Now we determine which individuals from the "best" island are chosen to migrate by spinning the roulette wheel. The roulette wheel chooses a number of migrants chosen according to their proportional fitness, we remove them from their island of origin (the best island), and add them to the destination island (the worst island). To maintain the size of the island of origin, we replace those fit migrants with randomly generated chromosomes.

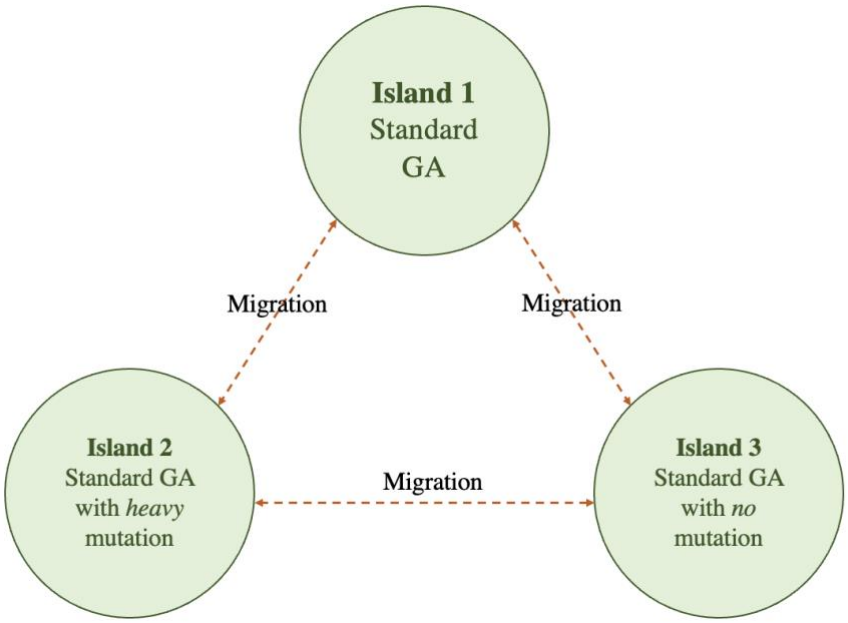

FIGURE 3. ISLAND GA TOPOLOGY CHOSEN FOR THIS PROJECT

SOME CONSTANTS REGARDING ALL EXPERIMENTS

For all the experiments, we initialize the population based on the domain on which the function of interest is defined, with the exception of x^2, which really exists from $(-\infty, \infty)$, but we constrained to [-10,10]. Each experiment is run for 50 generations and initializes a population of size 20. This gives us sufficient time and space to test out the algorithm while keeping the use of compute resources to a reasonable amount. Beyond this, parameters relating to roulette wheel, mutation, and elitism are changed and analyzed throughout the experiments.

EXPERIMENTS AND RESULTS

Now that the general algorithm for a GA is established as well as some specificities of implementation strategies, and we've narrowed in on the choices that will be made for this study, we can dive into the experiments. In this section, I show how I implement the GA algorithm first for x^2, and then for more difficult, higher dimension test functions chosen from [1]. Finally, I demonstrate how I use an island GA for one of the higher dimension test functions.

I. MINIMIZE X^2

The first task is to implement a GA to minimize $f(x) = x^2$. This is a trivial problem since this function is a single-variable problem that has exactly one minimum, but it is desirable as a starting point for its simplicity and understandability. As discussed in the Implementation section, we implemented the GA using roulette wheel and simplex crossover. We initialized the population on the domain [-10,10]. I then investigated using this algorithm with and without elitism.

The first run of this algorithm implements roulette wheel with the specification that a parent gets removed from consideration if he has already been selected twice. This prevents a single parent with a disproportionately high probability of selection from overrunning the population. The first run also has elitism turned off and has a high mutation rate, 0.9. Figure 4 shows a model of the idealized function with black dots representing the individuals in the population. By the fiftieth epoch, the function looks like it has been successfully minimized.

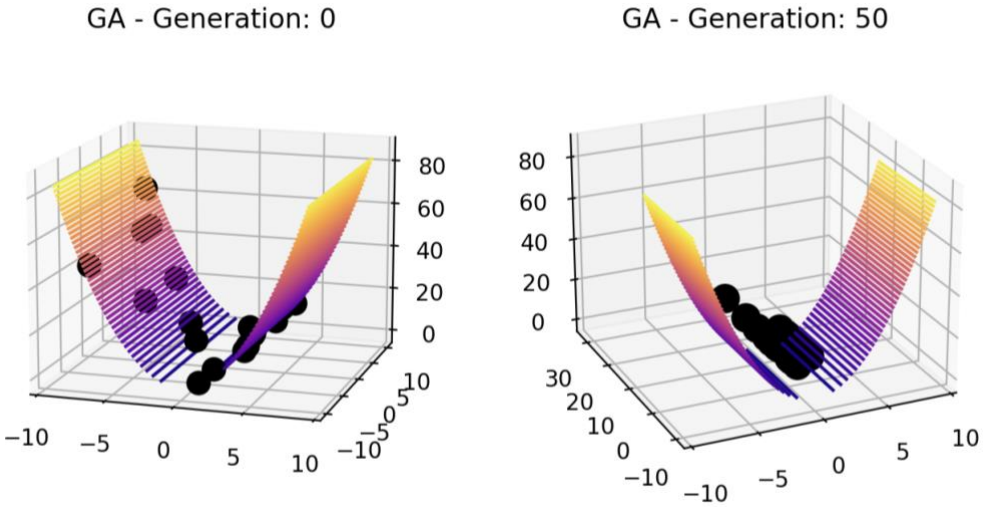

FIGURE 4. INITIAL POPULATION AND FINAL GENERATION FOR MINIMIZATION OF X^2

This looks quite ideal because the dots have migrated towards the minimum, but the Fitness vs. Generations graph (Figure 5) provides some additional, nuanced insight into what this algorithm is doing over time. Notice that we get "lucky" in our random initialization: We have immediately found the minimum, and since we are using simplex crossover, we are always averaging with this minimum value, so the minimum stays flat throughout the whole graph. Still, by the 20th generation we begin heavier exploration after a long period of near-zero exploration. So we begin to find these randomly high values as a result of mutation. The minimum remains quite low throughout the runtime, and the mean stays pretty close to the minimum. This

effectively “solves the problem,” but overall the algorithm has room for improvement. We want to have decent exploration throughout, not random jumps only beginning when we are half-way through the number of generations.

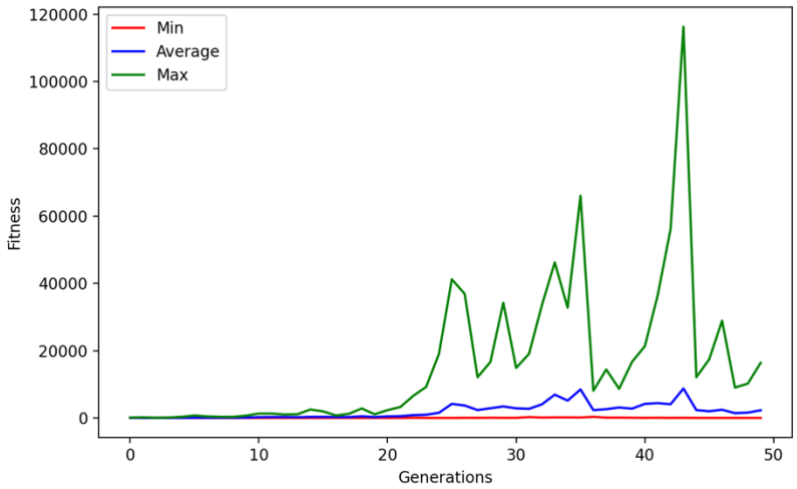

FIGURE 5. FITNESS VS GENERATIONS FOR THE MINIMIZATION OF X^2

To improve on this, we introduce elitism. Elitism takes out a little bit of the randomness by ensuring that the most fit will survive and reproduce. So with elitism turned on, we get a downward trend on the maximum (Figure 6), which is the opposite of what we see in Figure 5. This is good evidence that as more fit individuals are being chosen for reproduction, fewer unfit individuals are created over time. This is especially apparent from the initial huge spike early in the execution of the program, seen in Figure 6*. We also get higher diversity throughout the entire algorithm, which is desired. Interestingly, the visualization of the graph at generation 50 in Figure 6 looks a little worse than the one in Figure 4. The Fitness vs. Generations graph is crucial to helping us understand why this would be the case. Since a GA is a “do until done” algorithm, we stop at 50 generations, which is a somewhat arbitrary number. If we had stopped at just over 40 generations, the graph in Figure 6 would look great, and the graph in Figure 4 would look much worse. So we cannot put too much faith in the final generation; we cannot expect that this is *the* point at which the algorithm “converges.” The important thing is that the algorithm has had enough time to exploit and explore, and we see an overall trend toward the solution, which notably happens pretty well for the combination of simplex crossover and roulette wheel, both with and without elitism

*We should note the graph of x^2 in Figure 6 is shown in 3D space, even though this was really implemented for a 2D problem. This is because I started writing the paper after I extended my algorithm to two dimensions, and plotting this function in 3D space still provides a meaningful picture of minimization of the graph.

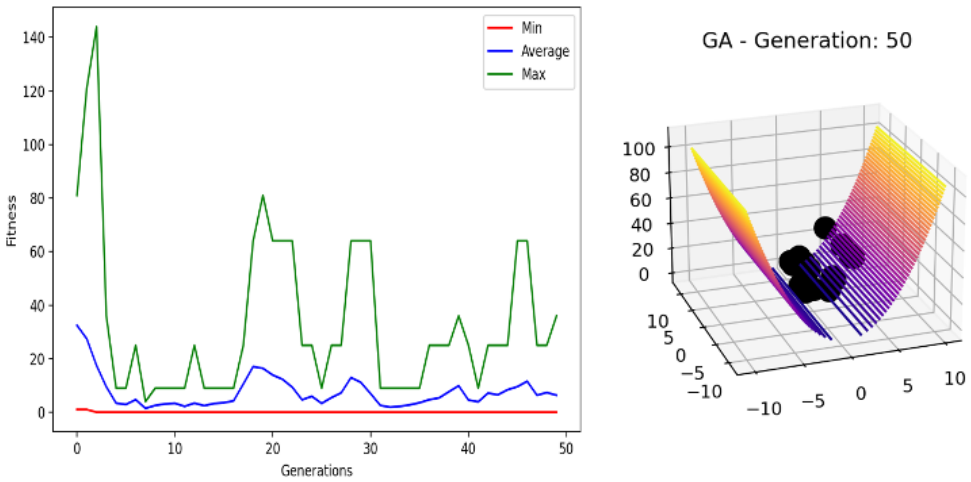

FIGURE 6. FITNESS VS GENERATIONS AND VISUALIZATION OF GENERATION 50 FOR THE MINIMIZATION OF X^2 WITH ELITISM TURNED ON

II. MINIMIZATION OF THE MATYAS FUNCTION

Now that we have dived into the details of implementation of a GA, we are ready to try a slightly more challenging function. The two-dimensional graph of the Matyas Function from [1] is shown in Figure 7. The Matyas function is defined as follows:

$$f(x,y) = 0.26(x^2 + y^2) - 0.48xy \quad -10 \leq x,y \leq 10$$

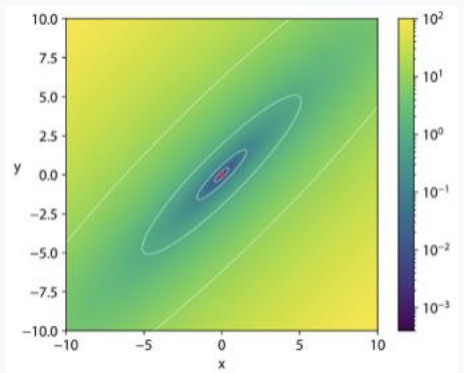

FIGURE 7. THE MATYAS FUNCTION [1]

Leaving elitism on, we observe how the algorithm performs for the minimization of this slightly more complicated, two-dimensional function with an initialization of the population from values between [-10,10]. The initialized population is shown next to the final iteration in Figure 8 superimposed over the idealized function in three-dimensional space. It looks like the algorithm found solutions close to the minimum, but as before, the Fitness vs. Generations graph (Figure 9) can provide additional insight.

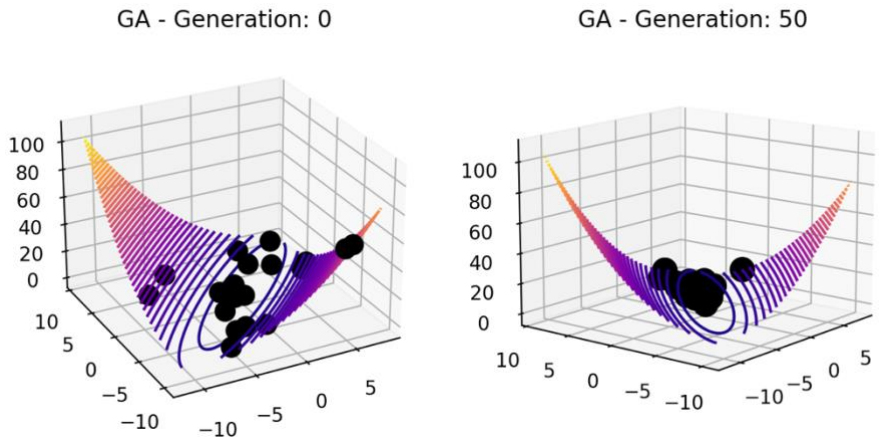

FIGURE 8. INITIAL POPULATION AND FINAL GENERATION FOR MINIMIZATION OF THE MATYAS FUNCTION

Figure 9 gives us a really interesting picture of the performance of this algorithm on the Matyas function. We see the initial population lucking out and finding the minimum immediately, and just as was the case with x^2, we might propose that elitism is pulling down the maxima, so we get this downward trend of the maximum curve and a flattening of the mean until about generation 40, when we find a large maximum, and then the max trends dramatically upward, giving the mean a little bit of a bump as well (although, due to the combination of elitism and simplex crossover, we still get a flat minimum at the solution). This behavior is likely due to high mutation mixed with roulette wheel, which, though constrained by setting a limitation on the number of times a parent can be selected, still encourages randomness.

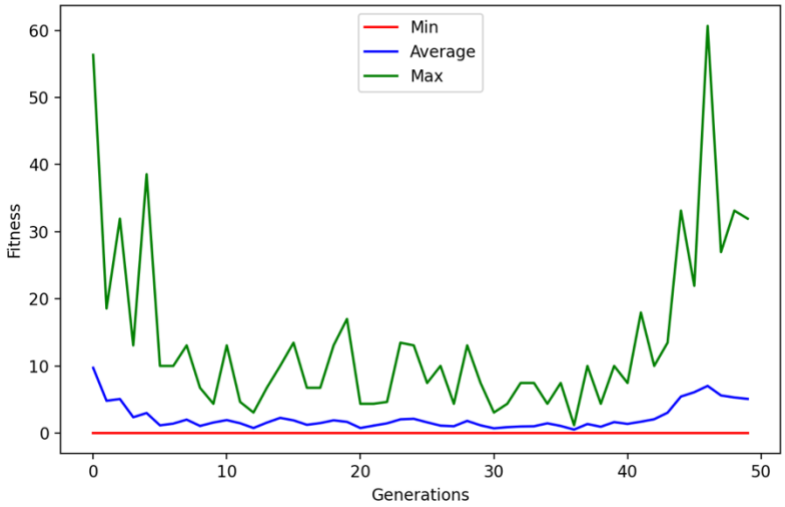

FIGURE 9. FITNESS VS GENERATIONS FOR THE MATYAS FUNCTION

III MINIMIZATION OF HIMMELBLAU'S FUNCTION

Himmelblau's function is interesting because unlike Matyas and x^2, which only have one minimum, it features four minima. The two-dimensional graph of Himmelblau's Function from [1] is shown in Figure 10, and the function is defined as follows:

$$f(x,y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2 \quad -5 \leq x, y \leq 5$$

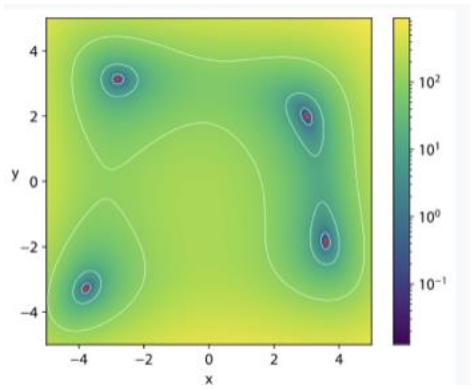

FIGURE 10. HIMMELBLAU'S FUNCTION

We initialized the population for values between [5,5] because this is the domain for which the function is defined. Figure 11 shows the three-dimensional graphs with the populations superimposed for the initial population and the population of the final generation. We can see the final generation concentrated on one of the four minima, but it is a little easier to visualize this looking at the two-dimensional cross section of Figure 12.

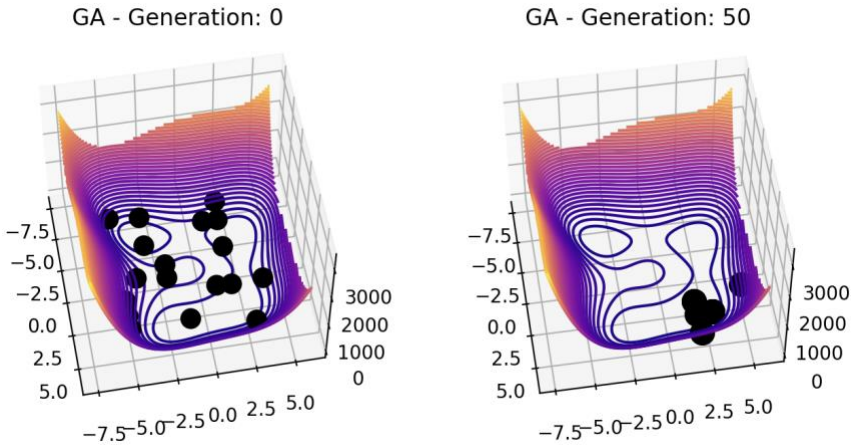

FIGURE 11. INITIAL POPULATION AND FINAL GENERATION FOR MINIMIZATION OF HIMMELBLAU'S FUNCTION

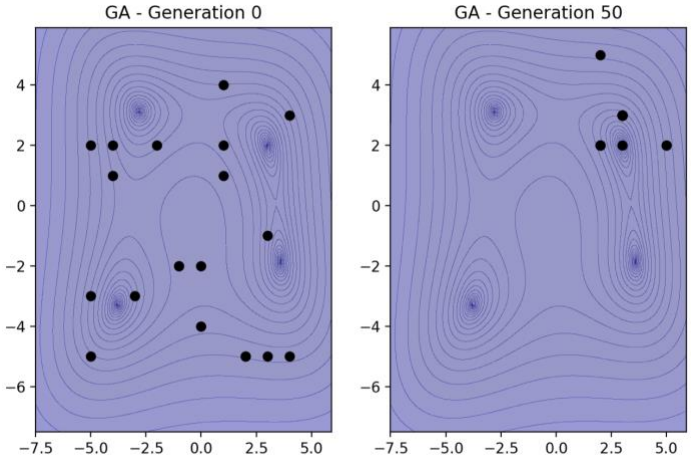

FIGURE 12. TWO-DIMENSIONAL CROSS SECTION OF INITIAL POPULATION AND FINAL GENERATION FOR MINIMIZATION OF HIMMELBLAU'S FUNCTION

Notice this function features one minimum in each quadrant. I find it interesting that every time I run my program, I consistently find the minimum at f(3.0, 2.0), the minimum in quadrant I. Figure 13 shows the outcomes of multiple executions of the program with slight variations on clustering around the same minimum. These results were obtained from running the program with varying levels of mutation, with and without elistism. If we force negative values by initializing the population with values between [-10,0], it finds the minimum at f(-3.78,-3.28), (see Figure 14), but when the full range of values are available, it consistently favors the only minimum with positive x and y values. I do not understand why the GA generally favors the first quadrant, but it is good to know that it is still able to find this minimum when given a nudge.

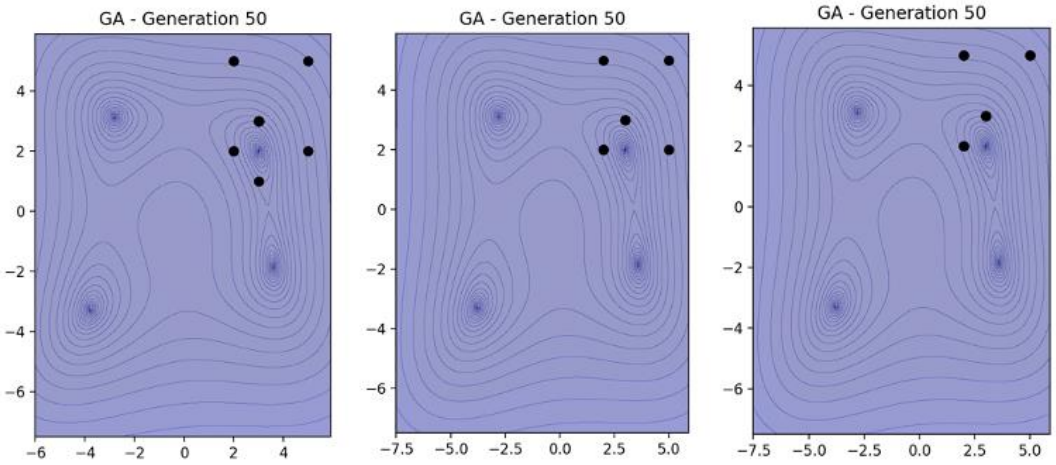

FIGURE 13. MULTIPLE EXECUTIONS OF THIS PROGRAM RESULT IN VARIATIONS ON CLUSTERING AROUND THE SAME MINIMUM AT F(3.0,2.0)

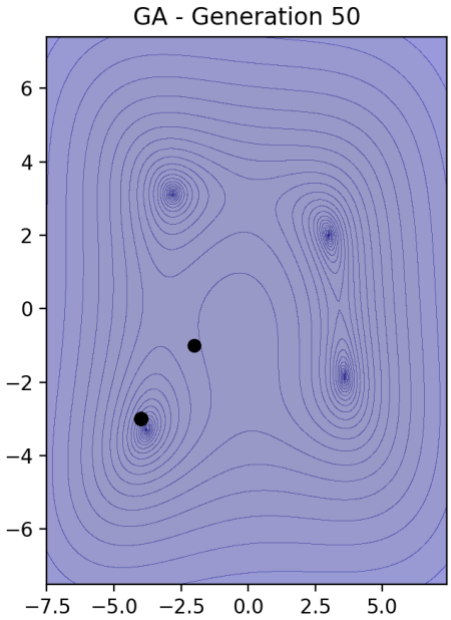

FIGURE 14. MINIMUM FOUND AT F(-3.78,-2.38) WHEN NO POSITIVE VALUES ARE INCLUDED IN THE INITIAL POPULATION

The Fitness vs. Generations graph (Figure 15) for this function gives us the same clues about the genetic algorithm that we got from previous applications. Some space between the max and the mean tells us that we are doing some exploring, though there is certainly room for more, and a flat minimum tells us we find a solution early and that the combination of elitism and simplex crossover keeps us from losing memory of that solution.

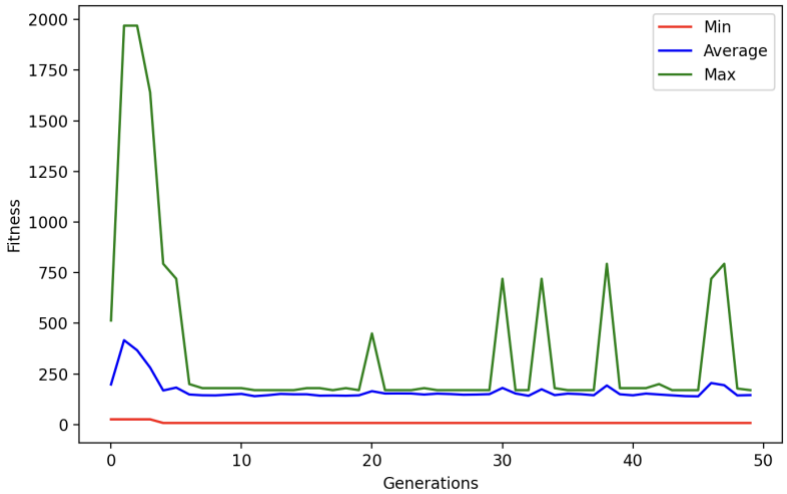

FIGURE 15. FITNESS VS GENERATIONS FOR HIMMELBLAU'S FUNCTION

III. MINIMIZATION OF THE ACKLEY FUNCTION

Perhaps a more complicated function with many local minima will tell us more about the performance of our genetic algorithm, especially with respect to this pattern of always quickly finding the minimum. The Ackley Function, visualized in Figure 16, is given by:

$$f(x,y) = -20exp\left[\sqrt{0.5(x^2+y^2)}\right] - exp[0.5(cos2\pi x + cos2\pi y)] + e + 20$$

$$-5 \leq x, y \leq 5$$

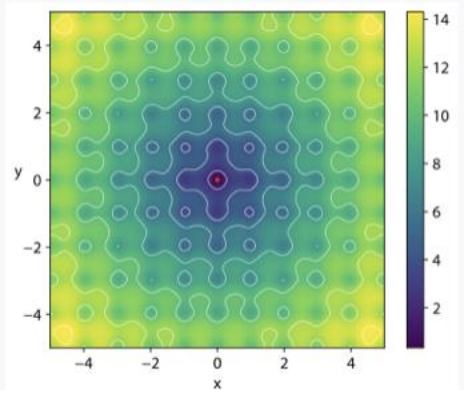

FIGURE 16. THE ACKLEY FUNCTION

This function has a lot of local minima, but only one global minimum, so we will have a much better idea of how robust our GA is if it can manage not to get stuck in a local minimum, but to find the global minimum. The initial population is shown next to the population of the final generation in Figure 17. It clearly clusters towards the global minimum with some variation. There is much more to do in this function compared to the others, so it is not surprising to see

significantly more variation in the Fitness vs Generations graph of Figure 18. There is more distance between the minimum and maximum curves in this graph that in any of our previous cases. Also, we find the solution pretty quickly, but not as quickly as with x^2, Matyas, or Himmelblau's. As we progress from simple to complex in terms of the fitness function, it is clear that the genetic algorithm is better suited to complicated problems than simple ones, although it is still capable of finding the solution to the simple ones.

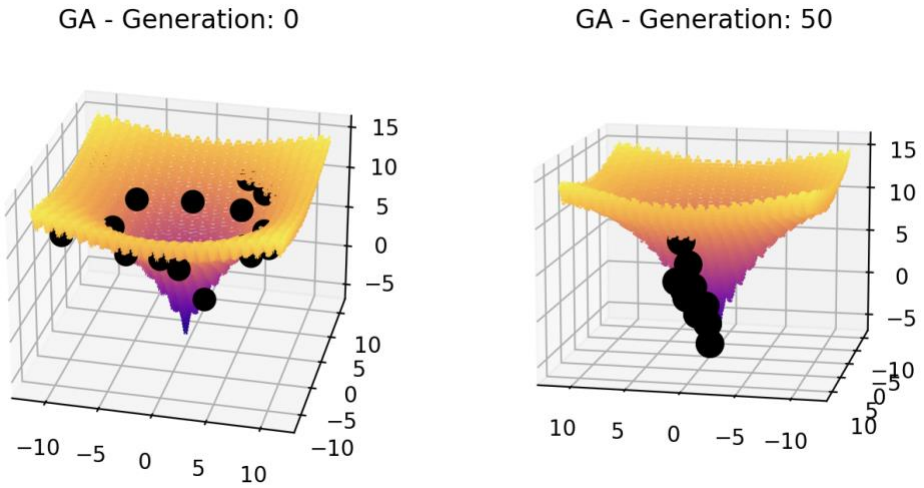

FIGURE 17. INITIAL POPULATION AND FINAL GENERATION FOR MINIMIZATION OF ACKLEY'S FUNCTION

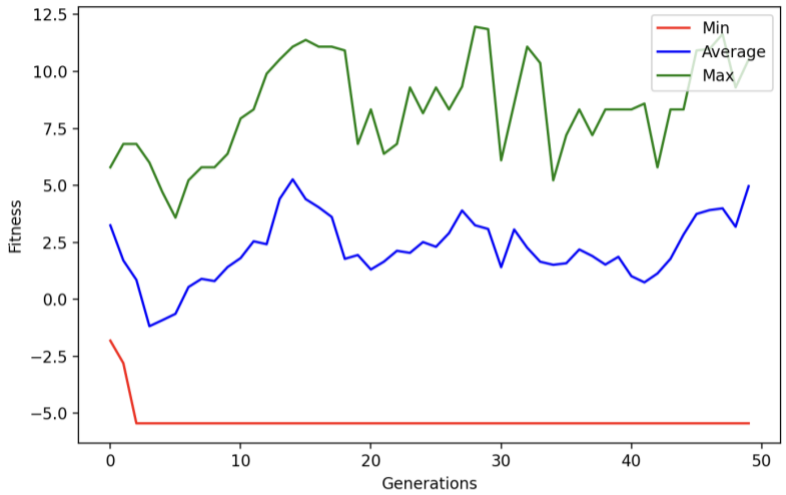

FIGURE 18. FITNESS VS GENERATIONS FOR ACKLEY'S FUNCTION

This causes us to wonder, if the combination of roulette wheel and simplex crossover are so good at balancing exploration and convergence on the solution, do we even need elitism? Well, the answer may well be yes; look at how poorly the algorithm performs when we turn elitism off for the Ackley function in Figure 19. We found the solution before 10 epochs, but we retained no memory of it without elitism, and roulette wheel wasn't strong enough to compete with such heavy mutation. Thus, we explored ourselves heavily away from the solution. Even when we change the mutation rate dramatically from 0.9 to 0.1, the graph looks about the same as that in Figure 19.

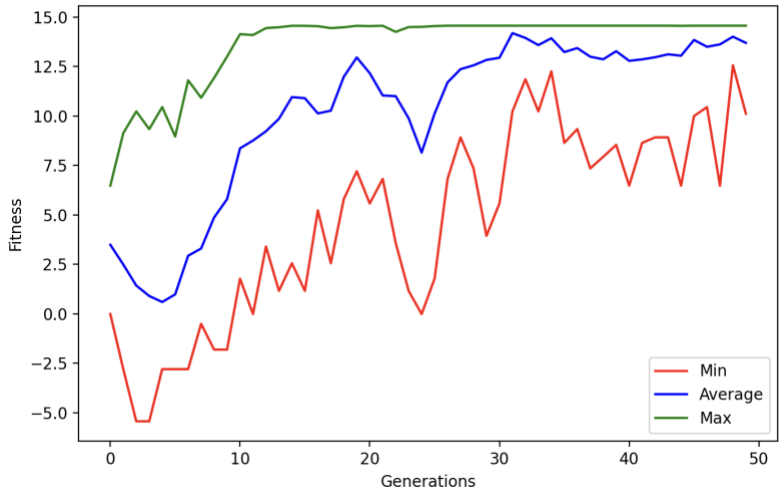

FIGURE 19. FITNESS VS GENERATIONS FOR HIMMELBLAU'S FUNCTION WITH ELITISM TURNED OFF

A final note on this: in the previous experiments, roulette wheel was written so that a unique parent could only be picked twice, so it was constrained, but somewhat conservatively. When we raise this number to allow four unique parents to be chosen and keep elistim off, we get the graph in Figure 20. This graph tells us that we are still better off with elitism—notice again we had the solution before 10 epochs and lost it—but roulette wheel can be tuned somewhat to imitate elitism, which is an interesting feature.

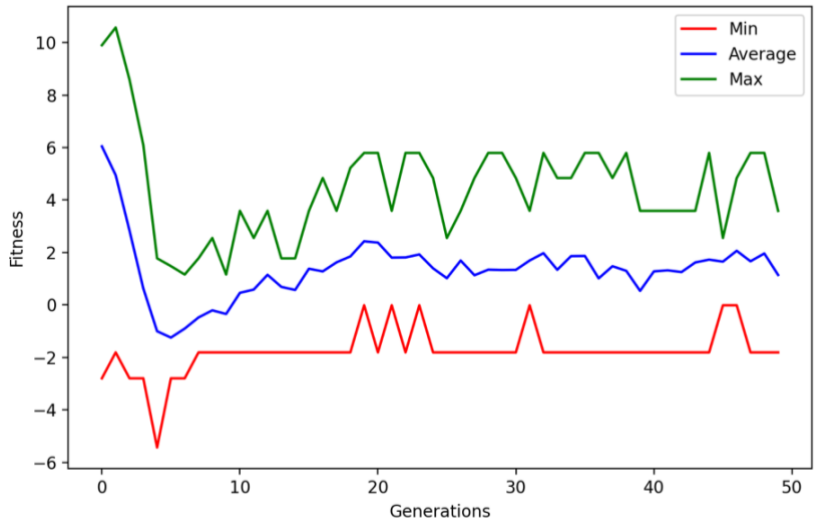

FIGURE 20. MINIMIZING ACKLEY'S FUNCTION WITHOUT ELITISM BUT WITH LESS CONSTRAINED ROULETTE WHEEL

IV. ISLAND GA TO MINIMIZE HIMMELBLAU'S FUNCTION

We enter into the Island GA portion of this project accompanied by a persisting curiosity about why the standard GA favors the solution at f(3.0,2.0) for Himmelblau's Function, when there is a solution in each quadrant of the space (Recall Figure 10, which shows these solutions). Perhaps we can leverage an Island GA to converge around one of the other solutions for this function.

With this in mind, we made the decision to completely move, rather than copy, the migrants from the "worst" island to the "best" island. Copying would have encouraged convergence, because we would have forced a greater overall fitness for all islands. Moving encourages exploration, because by replacing those fit individuals with randomly generated chromosomes, we maintain some diversity. Additionally, we build this island GA around mutation. If one GA mutates heavily while another is more focused on exploitation, perhaps that heavily mutating GA will get us to converge on a less-favored minima.

The first look at this implementation has each population initialized randomly on the entire domain. The Control Island is implemented with medium-mutation, the heavy-mutation Exploration island has both mutation rate and mutation scale set high, and the Exploitation island has mutation rate and scale set close to zero. For good measure, elitism is turned off for the exploration island, and it is set to 20% for the exploitation island. We observe the behavior of each island across the 50 generations in Figure 21. This graph is a little disappointing because the Exploration island actually explores less than the others after the 15th epoch or so, and yet it does not end up finding the solution. It just gets stuck at some minima near the solution, while the others explore around a little bit, but ultimately behave the way we've seen the standard GAs behave so far.

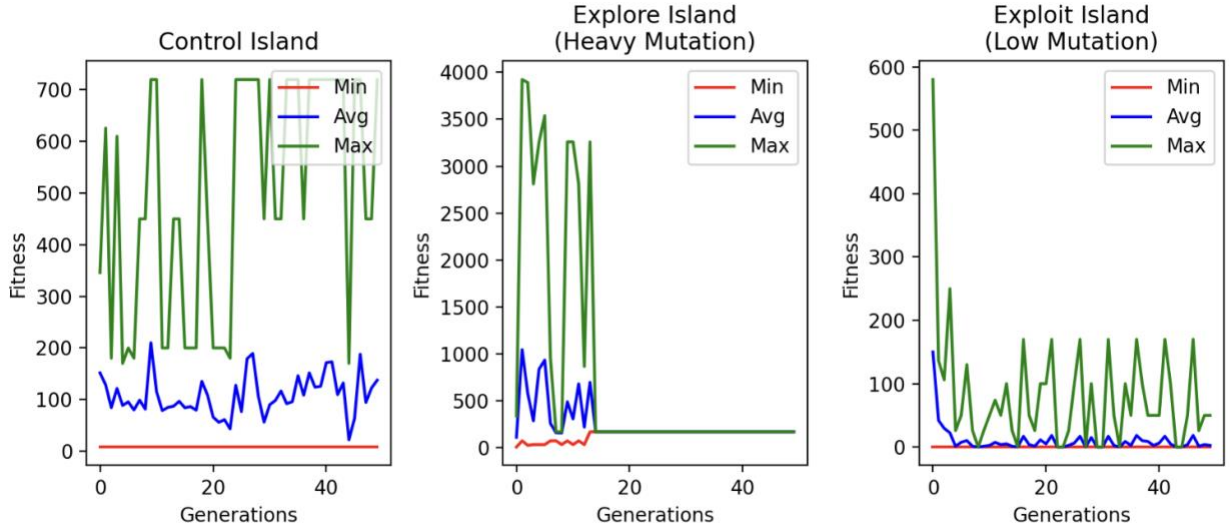

FIGURE 21. FITNESS VS. GENERATIONS FOR THE ISLAND GA ON HIMMELBLAU'S FUNCTION

We look for additional insight by graphing the solutions found by the final generation against the initial population. For Himmelblau's function, the two-dimensional graph is informative for identifying which minimum we converge around, shown in Figure 22. In some respects, this figure looks just about how we might expect it to look: the control and exploitation islands both find a minimum, with the exploitation island converging more tightly around the solution than the control island. The exploration island does not find the solution, which makes sense; heavy mutation means a lot of randomness. Notice, though, how the control island finds

the minimum in quadrant III! This is such an interesting, and really delightful, discovery! When we ran the standard GA multiple times, making various changes to cover the range of mutation, we always converged on the quadrant I solution unless we forced the domain to negative values.

How does the island GA achieve this? I really do not know, because I am puzzled as to why the single GA so heavily favors the one solution. With that being said, it does not feel surprising that it was the control island which found the previously elusive solution. The explore and exploit islands were so heavily built to do one task. The control island has balance between exploration and exploitation, but the thing that sets it apart from the single island (which, recall, is configured the same way as the control island) is that it receives genetic material from the heavily modularized island GAs. Just enough, it seems, to push it towards a different minimum. It seems this is the beauty of the island GA: the ability to modularize genetic algorithms and introduce genetic material in such a way that allows for both heavy exploration and heavy convergence.

This is by no means a perfect implementation: We still have the somewhat puzzling problem of strangely early convergence of the exploration island. But the fact that we found an unfavored minimum and see a decent amount of diversity in the control island is promising.

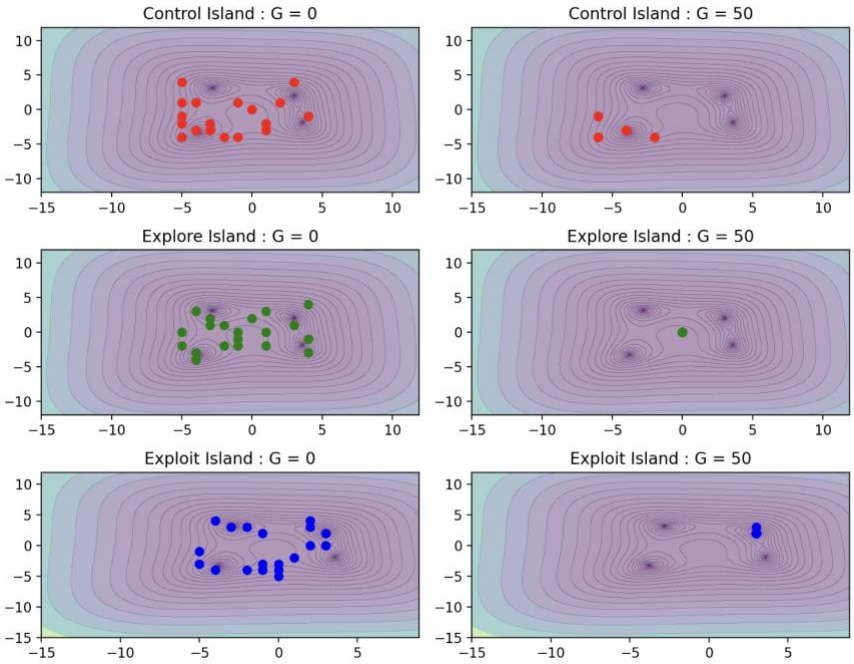

FIGURE 22. TWO-DIMENSIONAL CROSS SECTION OF INITIAL POPULATION AND FINAL GENERATION FOR MINIMIZATION OF HIMMELBLAU'S FUNCTION FOR ISLAND GA

SUMMARY OF RESULTS AND CONCLUSION

These results show that genetic algorithms are good at solving easy problems. The minimization of x^2 and the small jump to the Matyas function gave us an opportunity to build confidence that a basic GA will find the minimum starting from a population of random individuals. Further, these results show that the more complicated the problem, the more interesting it becomes to tune the algorithm. Problems like Himmelblau's function and the multi-minima Ackley function highlight the benefit of heavy exploration, and they raise interesting questions about how to balance randomness—which gives us exploration—with a kind of conservatism we get from elitism—which encourages the algorithm to converge on some solution. We don't want to be too conservative, lest we converge on the wrong solution. But we also don't want to be too unconstrained, as we saw when we turned elitism off for Ackley's function, such that we explore ourselves away from the solution. This balancing act changes for each question we might ask; i.e. in our case, for each function we might try to minimize. This brings up the important "no free lunch" theorem, which says there is no one algorithm—not even any particular implementation of an algorithm—that is the superior method for solving all problems.

With this in mind, it seems that the island GA is better-suited than a single GA for solving the minimization of Himmelblau's function. The more complicated the problem, and the more frequent the possible solutions, the greater the need for exploration. The island GA we implemented showed us that we could not only match the single GA's ability to achieve convergence, but we could also solve a deeper problem of varying the solution we find.

The goals of this project were to gain a meaningful understanding of GAs, to grow in the ability to implement and tune them, to be able to evaluate their performance, and ultimately to be able to formulate a question about how a GA might help me solve my research questions in the field of electromagnetics and the design of optical metamaterials. On completing this work, I am proficient in the understanding, implementation, and tuning of the basic algorithm. The island GA gave me the opportunity to gain a deeper understanding of and build on the foundation of GAs, especially the concept of mutation. I am still considering how I might make use of a genetic algorithm in my research, but this does not worry me. I am in the early stages of my research, still building the broader foundation of knowledge that I know will lead me to exciting questions. I look forward to the opportunity to use a GA to solve a research problem someday.

FUTURE WORK

There is an abundance of future work to consider following completion of this project! Because genetic algorithms give us so many options, we left many well-understood operators unexplored; for example, we did not attempt to implement tournament or rank selection, we did not look at k-point crossover or consider all the different ways we could add nuance to our implementation of mutation. The text describes a method of mutation by which we actually update the mutation rate with each generation such that we explore heavily in the beginning and much more conservatively toward the end; this method would have been very interesting to investigate [2].

Besides that, there are also plenty of ways we could improve or learn more about the algorithm we built. We demonstrated that our implementation of simplex crossover is working well to encourage convergence, but we might find a way to improve it since we use randomness in our aggregation of chromosomes. This is also true for roulette wheel, where we choose a random number between 0 and the maximum value to make our selection. It would be interesting to make a decision about these values based on fitness rather than randomness, partly because we could investigate how it encourages smart exploitation, and partly because it would allow us to delegate random exploration more fully to the mutation task. This might give us more confidence about what is happening overall in our algorithm, and thus give us more freedom and confidence in our ability to tune it. Still, this is a gray area. Some of the power of simplex crossover may be in its introduction of probability which gets you closer to convergence while maintaining some variation, whereas k-point crossover's arbitrary swapping of genes has variation built in.

Finally, the question persists: Why does a single GA prefer the minimum in quadrant 1 of Himmelblau's function? How do we even approach this question? It was gratifying to watch the island GA give us a different result, but it did not find us an *answer*, it just found the solution. In other words, there was nothing "explainable" about this result. Perhaps the answer to this question does not lie in the implementation of a GA, but elsewhere, or maybe it is just a mystery.

REFERENCES

[1] "Test functions for optimization," *Wikipedia*, Jan. 28, 2020. https://en.wikipedia.org/wiki/Test_functions_for_optimization

[2] A. P. Engelbrecht, *COMPUTATIONAL INTELLIGENCE : An Introduction.* S.L.: John Wiley, 2020.