University of Missouri

Project 1: Evolutionary Algorithms

Student: Aaron Harrison

Student ID: 18185071

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF EQUATIONS

1 Introduction

1.1 What is an Evolutionary Algorithm and What is its Connection to Nature?

In simple terms an evolutionary algorithm is an optimization program inspired by the biological term evolution. Evolution is the optimization process of an organism or system to survive in dynamic and competitive environments. An evolutionary algorithm more specifically relies on the term natural selection purposed by Charles Darwin in 1859. Natural selection has two main components: survival of the fittest and randomness. Survival of the fittest is where the best suited to survive in a specific environment repopulate and propel the population forwards. Randomness is the sense that sometimes an organism not suited to survive manages to reproduce as well. A common example of evolution is Darwin's Finches.

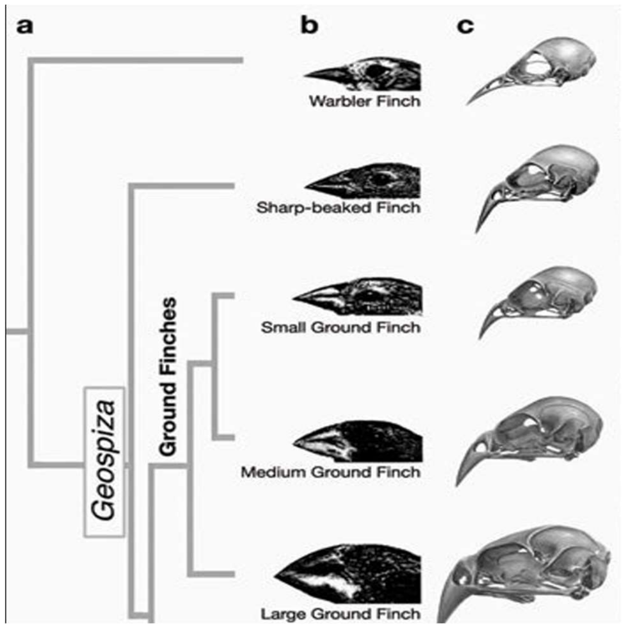


Figure 1 A depiction of Darwin's Finches

In Darwin's Finches each species of finch started from a common ancestor. Each species moved to a different environment, and therefore evolved differently, most evident in their beaks. With each new generation they became more suited to their environment than the previous generation. Thus, also changing more and more from the starting common ancestor. How did these animals evolve generation after generation though?

Every animal is made up of DNA which is responsible for making an animal what it is. Segments of DNA are called genes and they are responsible for specific traits of an animal. The genotype of an animal is how the DNA is made up and the raw structure of the gene. Whereas the phenotype is the actual expression of a gene. Think of a person being tall; the random sequence of genes is the genotype, but the expression of being tall is the phenotype. DNA is stored on chromosomes with each chromosome containing unique genes.

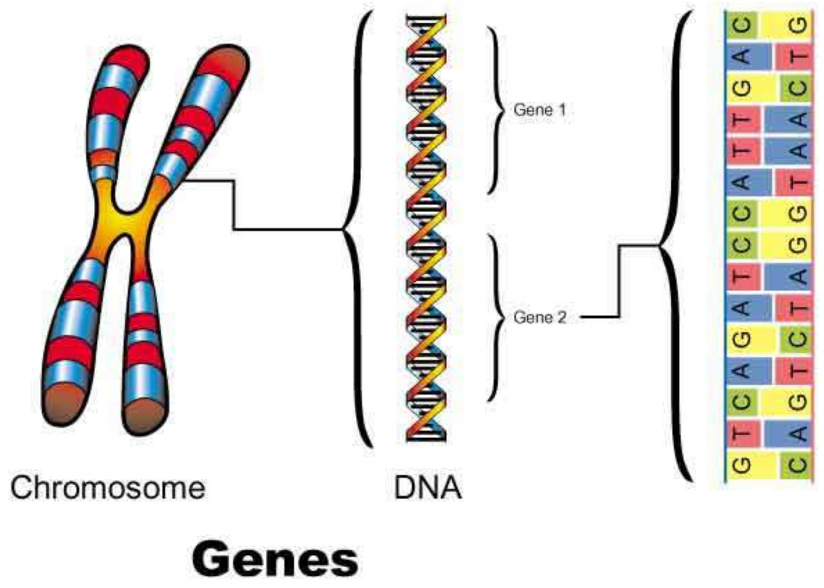


Figure 2 Illustration of a Chromosome, DNA, and Genes

Whenever two members of a species mate through sexual reproduction there is a crossover in chromosomes. More simply put, some genes in the chromosomes are received from the mother and some are received from the father. In nature, many species will mate based on who is most opportune for survival. Either because this is what attracts other members, or the ones not fit for survival have passed away before this stage. When this happens the genes that are being passed on are the ones that are most optimal for survival and therefore you see survival of the fittest but in the since of genes.

There are times where less fit members will reproduce, and this adds an element of randomness. Another item that adds randomness are recessive genes, but that is not included in the scope of genetic algorithms. Lastly, randomness can be added through mutations. Which is where a gene will change from both its mother's and father's genes to something completely new. However, mutations are relatively rare and will only affect a small amount of the population per generation.

A genetic algorithm simplifies this system to create an optimized search algorithm to find what values are the most fit for a certain problem. Another important part of a genetic algorithm is the concept of Exploration vs Exploitation. Exploration is the diversity of a certain group. Exploitation is the local search for the best.

1.2 General Structure of a Genetic Algorithm

Below is the general structure of a genetic algorithm:

```
Let $t = 0$ where $t$ is the generation counter
Initialize Population
While stopping conditions not true:
    Evaluate Fitness
    Selection based on fitness.
    Reproduction
    Mutation
    $t = t + 1$
```

1.3 Initialization

What does a chromosome look like in terms of a genetic algorithm, and where does the first set of them come from? In most cases a chromosome will be an array that contains values. Each value can be thought of as a gene. These genes will refer to specific parameters of our problem. Each population is made up of a set of chromosomes, and every individual of the population will only have one chromosome. The notations that will be used for chromosomes will be:

$$X_i(n) \in \mathbb{R}^k = [p_1, p_2, \dots, p_k]$$

Equation 1 Notation of Chromosome

Where X is the set of chromosomes, i is the current generation, n is the specific chromosome in the ith generation, and $\in \mathbb{R}^k$ means that there are k parameters (genes) and each of them is in a real feature space.

There are many ways to initialize a starting population, but the main goal is to have good diversity, so the algorithm is searching the entire space. The two ways that are implemented in this paper are random and grid. They both do not take in any prior knowledge to a search space but have good diversity.

1.3.1 Random Implementation

The random implementation is quite simple, each parameter of every chromosome is chosen randomly within the search space with no knowledge of the previous chromosomes or parameters.

$$for\ all\ n\ and\ for\ all\ p{:}\ X_0(n,p) \in \mho[min, max]$$

Equation 2 Random Implementation

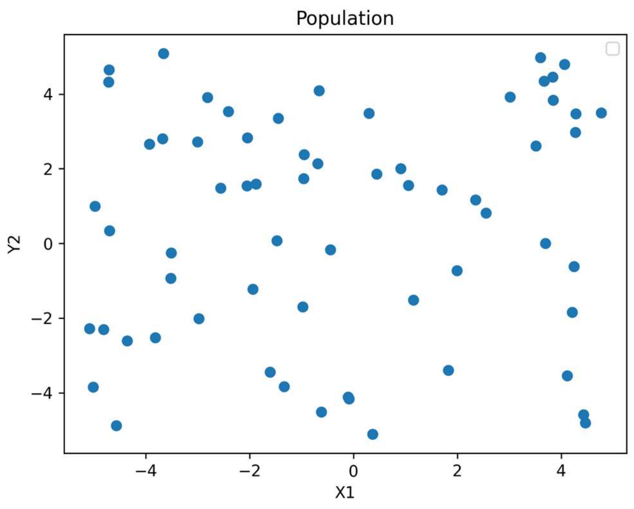


Figure 3 A randomly initialized population

1.3.2 Grid Implementation

The grid implementation is simple to understand, but the implementation gets complicated with varying dimensions. It uniformly distributed the chromosomes throughout the entire search space.

The first step is to calculate the space needed between each point for each dimension.

$$space_p = \frac{\max - min}{population\_size_p - 1}$$

Equation 3 Calculate space for grid initiation.

Population_size$_p$ is the number of unique values in dimension p. Then all values of the first dimension are calculated:

$$for\ i\ in\ Population\_size_0 : X_0(i,0) = X_0(i-1,0) + space_0$$

Equation 4 Calculate the first dimension for grid initiation.

Each possible unique value from the first dimension is added to the population. Once the first dimension is created, each unique value from the second dimension is appended to each unique value from the first dimension. Each value from the third dimension is appended to each unique combination of the first and second dimension. This continues until each dimension is accounted for. The algorithm is as follows:

```
Population = [[]]
for p in dimensions
    new_population = [ ]
    for chromo in population
        for j = min; j<=max; j += $space_p$
            new _population.append(chromo.append([j]))
        population = new_population
```

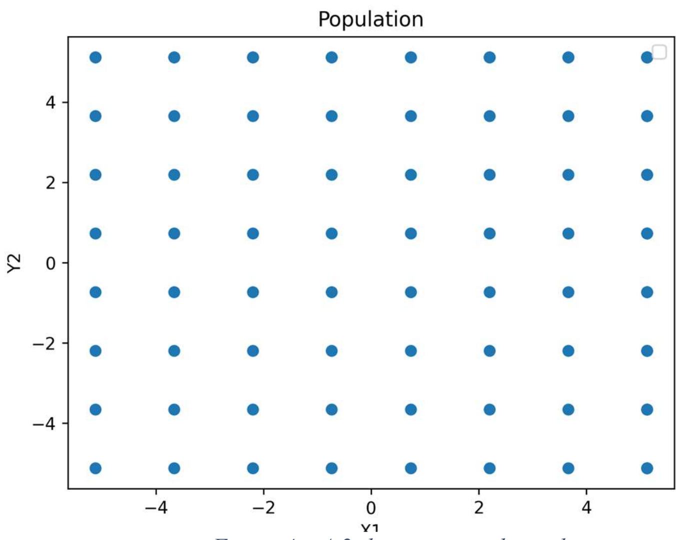


Figure 4 A 2-dimension grid population

1.4 Selection

Selection is the process of choosing the individuals that will reproduce to create the next generation. The selection process can be described as random with heuristics. This is done by evaluating the fitness of the individuals. Then choosing randomly, but with weight added depending on fitness. The fitness functions used in this paper are all minimization problems. They are the

Sphere:

$$f(x) = \sum_{i=1}^{n} {x_i}^2$$

$$global\min\ f(0,\dots,0) = 0\,, \qquad search\ domain - \infty \leq x_i \leq \infty$$

Equation 5 Sphere fitness function

Rastrigin:

$$f(x) = An + \sum_{i=1}^{n} [{x_i}^2 - Acos(2\pi x_i)]$$

$$where\ A = 10, \qquad global\min\ f(0,\dots,0) = 0\,, \qquad search\ domain - 5.12 \leq x_i \leq 5.12$$

Equation 6 Rastrigin fitness function

Rosenbrock:

$$f(x) = An + \sum_{i=1}^{n} [100(x_{i+1} - {x_i}^2)^2 + (1 - x_i)^2]$$

$$where\ A = 10, \qquad global \min\ f(0, \dots, 0) = 0\,, \qquad search\ domain -5.12 \leq x_i \leq 5.12$$

Equation 7 Rosenbrock fitness function

The implementations for selection are roulette wheel selection and tournament selection.

1.4.1 Roulette Wheel Selection

Roulette wheel selection is a method based off a roulette wheel, but with the main difference being the percentage chance the “ball” lands on a chromosome’s “number” is correlated to its fitness score. To calculate the percentage chance, you use:

$$p_s\big(x_i(t)\big) = \frac{f_y(x_i(t))}{\sum_{j=1}^{n} f_y(x_i(t))}$$

Equation 8 Roulette Wheel Selection

Where $p_s(x_i(t))$ is the percentage chance of a chromosome i in the current generation t, $f_y(x_i(t))$ is the fitness function of the individual. In simple terms it is the fitness of the individual divided by the total fitness of the population. It is implemented as:

```
Let $i = 1$
Sum = $p_s(x_i=1)$
Choose r to be random number between [0,1]
While (sum < r)
    $i = i + 1$
    Sum += $p_s(x_i)$
Return $x_i$
```

1.4.2 Tournament Selection

Tournament selection is the method of randomly selecting a certain number of individuals and the highest fitness score moves on and the rest go back into the tournament pool. Once the population is the right size all the winners are the population, and the losers are discarded. It is implemented as:

Select n_{ts} – number of individuals to select

Compare performance

Best individual selected

The rest go back into the tournament pool

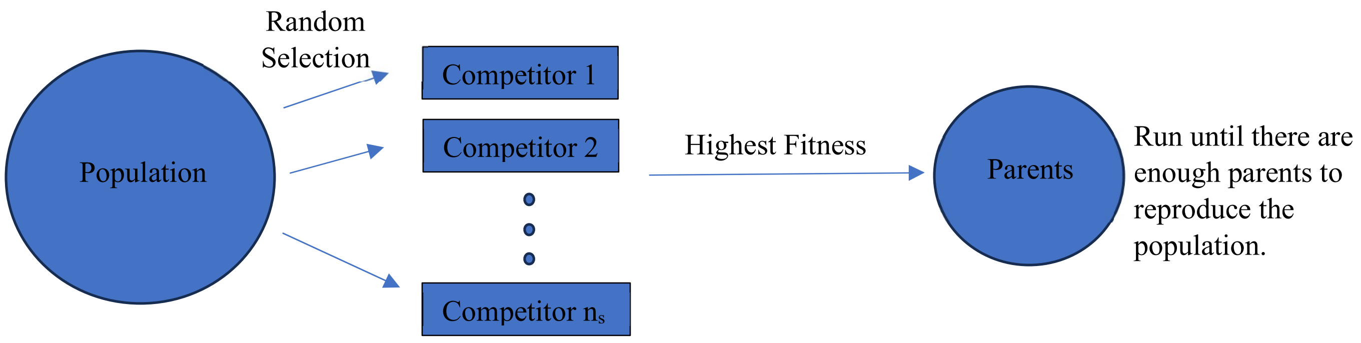


Figure 5 Diagram of Tournament Selection

1.5 Crossover

Crossover is the process of repopulating the population. This paper uses two methods of multiple parent reproduction. This process generates new unique chromosomes that have the foundations of the parents that were selected in the selection process. The two methods used are two point and SPX.

1.5.1 Elitism

Elitism is the process of passing on the most fit individuals to the next generation. In the implementation the top 5 members of the population will be passed on.

1.5.2 Two Point Crossover

Two points cross over randomly finds two values within 0 and the number of dimensions. For this explanation point 1 will be the smaller of the two and point 2 the larger. Two chromosomes are then randomly selected from the population and create two children. Their child is from dimensions 0 to point 1 minus one are the first chromosome values, dimensions from point 1 to point 2 are the second chromosome values, and dimensions from point 2 plus one the first chromosomes values.

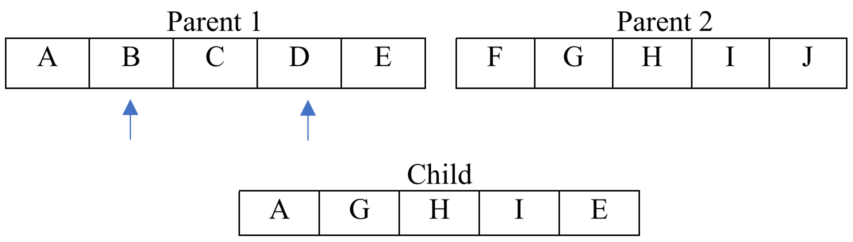


Figure 6 Example of two-point crossover

1.5.3 SPX (Simplex Crossover)

Simplex crossover is the process of forming a simplex between three points. In a 2d form the simplex is a triangle. Then a point is randomly found in the triangle and that is the child. Here the vertexes are computed as follows.

$$O = \frac{1}{3}\sum_{i=1}^{3} X_i$$

$$Vertex(Y_i) = (1+\varepsilon)(X_i - O)$$

Equation 9 Computation of Simplex Vertices

With ε as a control parameter determining how far outside the points is searched.

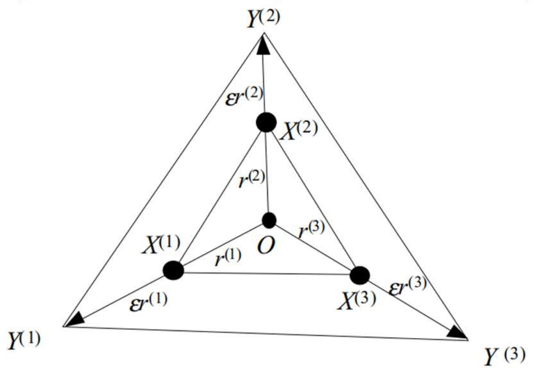


Figure 7 Representation of simplex computed in SPX

1.6 MUTATION

Mutation is the process of a property/gene having its value changed. The process is done by coming up with a mutation rate. These are normally relatively low. Then mutation is done on that percentage of genes out of the whole population. To do this a general structure is implemented of:

```
For every chromosome in population
    For every gene in the chromosome
        $R \in \mho[0,1]$ select random number between 0 and 1
        If R is less than the mutation rate
            Perform mutation on the gene
```

The two types of mutation used are uniform random and gaussian mutation.

1.6.1 Uniform Random Mutation

In uniform random mutation a variable is chosen as length. Where length is the furthest point a mutation can take place away from the current value. Then a random value is calculated between the current value - length and the current value + length. If this is past the bounds, then the value is set to the bound. The percentage chance for any variable between current value - length and current value + length is the same.

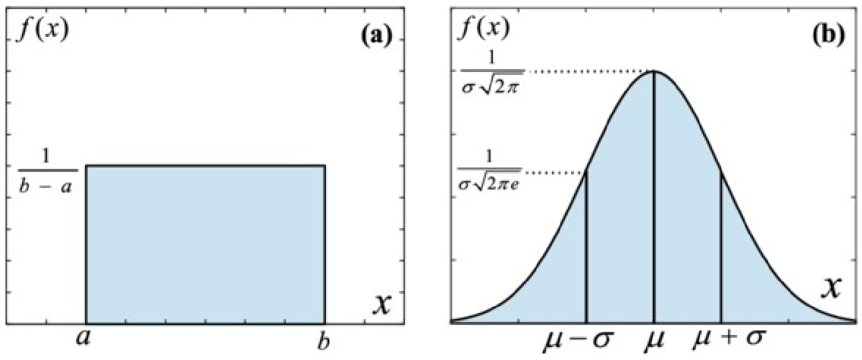


Figure 8 (a) is the percentage chance of the mutated value for uniform random. (b) is the percentage chance of the mutated value for gaussian mutation.

1.6.2 Gaussian Mutation

Gaussian mutation is based off the gaussian distribution. The current value of the property/gene is the center of the distribution. The standard deviation is passed in and this determines the likely hood of a certain value being calculated. 68% of the time the mutated gene value will come within 1 standard deviation, 95% of the time the value will be within 2 standard deviations, and 99.7% of the time the value will be within 3 standard deviations. This means that the mutated gene value is more likely to be closer to the original value but can still move further away on occasion.

$$\mathbb{N}(non\ mutated\ value, \sigma)$$

Equation 10 The notation for the gaussian distribution to sample from

1.7 TERMINATION

Termination occurs whenever the program should stop. There are many different criteria that can be met to stop a program. A fixed number of generations, which is done in this paper, stops after a determined number of generations have occurred. Fitness threshold termination, also done in this paper, stops whenever a certain fitness score has been attained. Convergence is when the population has developed into a state that is not changing.

2 EXPERIMENTS AND RESULTS

2.1 HIGH LEVEL COMPARISON OF GENETIC ALGORITHMS

For the high-level comparison of the genetic algorithms, the population size is 144, the fixed termination will have 100 iterations, the fitness termination will be a fitness below 0.09, the mutation rate will be 0.03, the standard deviation of gaussian mutation will be 0.6, and the uniform random mutation length will be 1. The data displayed will be the average and the standard deviation displayed as average(standard deviation).

Genetic Algorithm (Initialization, Selection, Crossover, mutation)	**Sphere**	**Rastrigin**	**Rosenbrock**
GA1 = Random, RWS, Two Point, Random, Fixed	0.056(0.304)	0.497(3.33)	2.06(18.6)
GA2 = Random, RWS, Two Point, Random, Fitness	8.12(16.8)	0.589(2.09)	2.38(17.9)
GA3 = Random, RWS, Two Point, Gaussian, Fixed	0.051(0.450)	0.500(2.31)	2.86(22.7)
GA4 = Random, RWS, Two Point, Gaussian, Fitness	1.71(2.25)	1.20(3.99)	3.44(21.8)
GA5 = Random, RWS, SPX, Random, Fixed	0.030(0.289)	0.944(3.43)	3.45(16.8)
GA6 = Random, RWS, SPX, Random, Fitness	0.443(0.823)	1.53(3.64)	19.38(30.5)
GA7 = Random, RWS, SPX, Gaussian, Fixed	0.067(0.479)	1.33(5.14)	3.23(19.7)
GA8 = Random, RWS, SPX, Gaussian, Fitness	0.247(0.414)	1.78(4.15)	2.52(9.72)
GA9 = Random, Tournament, Two Point, Random, Fixed	0.045(0.237)	0.287(1.92)	6.95(30.3)
GA10 = Random, Tournament, Two Point, Random, Fitness	21.4(19.1)	0.946(3.39)	3.86(18.3)
GA11 = Random, Tournament, Two Point, Gaussian, Fixed	0.037(0.140)	0.764(3.41)	3.19(16.1)
GA12 = Random, Tournament, Two Point, Gaussian, Fitness	66.1(41.7)	1.03(2.75)	2.89(18.0)
GA13 = Random, Tournament, SPX, Random, Fixed	0.015(0.108)	0.908(3.70)	0.667(2.84)
GA14 = Random, Tournament, SPX, Random, Fitness	2.41(2.57)	2.42(4.79)	52.7(42.85)
GA15 = Random, Tournament, SPX, Gaussian, Fixed	0.614(3.08)	2.06(10.2)	0.614(3.08)
GA16 = Random, Tournament, SPX, Gaussian, Fitness	100(119.3)	18.1(9.97)	10.1(17.2)
GA17 = Grid, RWS, Two Point, Random, Fixed	0.405(0.168)	1.47(2.51)	1.64(9.69)
GA18 = Grid, RWS, Two Point, Random, Fitness	0.994(0.638)	0.665(2.83)	2.46(13.0)
GA19 = Grid, RWS, Two Point, Gaussian, Fixed	0.077(0.379)	0.814(3.58)	2.06(10.2)
GA20 = Grid, RWS, Two Point, Gaussian, Fitness	1.37(1.86)	0.690(2.72)	1.92(8.76)
GA21 = Grid, RWS, SPX, Random, Fixed	0.059(0.336)	1.597(4.75)	3.52(12.7)
GA22 = Grid, RWS, SPX, Random, Fitness	71.1(850)	15.12(10.89)	2.40(13.1)
GA23 = Grid, RWS, SPX, Gaussian, Fixed	0.012(0.617)	0.780(3.62)	7.23(30.6)
GA24 = Grid, RWS, SPX, Gaussian, Fitness	0.938(0.905)	9.16(9.68)	1.41(7.30)
GA25 = Grid, Tournament, Two Point, Random, Fixed	0.076(0.391)	0.480(2.47)	2.36(13.0)

GA26 = Grid, Tournament, Two Point, Random, Fitness	0.784(0.423)	1.49(3.21)	2.27(9.84)
GA27 = Grid, Tournament, Two Point, Gaussian, Fixed	0.022(0.137)	0.581(2.70)	2.54(14.0)
GA28 = Grid, Tournament, Two Point, Gaussian, Fitness	0.754(0.589)	1.99(3.23)	2.13(16.2)
GA29 = Grid, Tournament, SPX, Random, Fixed	0.035(0.304)	0.482(3.17)	4.92(21.3)
GA30 = Grid, Tournament, SPX, Random, Fitness	24080(26037)	2.09(4.42)	1.90(10.2)
GA31 = Grid, Tournament, SPX, Gaussian, Fixed	0.065(0.309)	0.831(3.56)	2.04(9.61)
GA32 = Grid, Tournament, SPX, Gaussian, Fitness	27559(27150)	2.80(5.41)	3.20(20.9)

Table 1 Comparison of averages and standard deviations of Genetic Algorithms

From the table above a few things stand out. One is that the fitness termination may be ending well before the population has converged on occasion. This could be due to the termination occurring once a certain fitness score has been achieved. It is possible that one chromosome gets below the fitness minimum before the population has converged. Another item that can be seen from the table is that the sphere has the lowest averages and standard deviations, with the Rastrigin in second, and the Rosenbrock with the highest. This makes sense due to the complexity of the problems.

Genetic Algorithm	**Minimum Value**	**Maximum Value**	**Average Value**	**Standard Deviation**	**Time to Run**	**Number of Generations**
GA1 = Random, RWS, Two Point, Random, Fixed	0.135	19.8	0.497	3.33	2.32	100
GA2 = Random, RWS, Two Point, Random, Fitness	0.00367	20.2	0.589	2.09	0.245	14
GA3 = Random, RWS, Two Point, Gaussian, Fixed	0.0010	20.2	0.500	2.31	1.4	100
GA4 = Random, RWS, Two Point, Gaussian, Fitness	0.044	23.0	1.20	3.99	0.222	11
GA5 = Random, RWS, SPX, Random, Fixed	$8.99*10^{-7}$	15.4	0.500	2.31	2.10	100
GA6 = Random, RWS, SPX, Random, Fitness	0.0339	39.3	14.7	10.5	0.047	3
GA7 = Random, RWS, SPX, Gaussian, Fixed	$1.9*10^{-10}$	17.7	0.713	3.43	3.56	100
GA8 = Random, RWS, SPX, Gaussian, Fitness	0.001	25.0	15.74	10.17	0.050	2

GA9 = Random, Tournament, Two Point, Random, Fixed	0.00013	18.47	0.287	1.92	1.20	100
GA10 = Random, Tournament, Two Point, Random, Fitness	0.0875	21.8	0.946	3.39	0.263	16
GA11 = Random, Tournament, Two Point, Gaussian, Fixed	0.000212	19.8	0.764	3.41	1.04	100
GA12 = Random, Tournament, Two Point, Gaussian, Fitness	0.0350	10.3	1.03	2.75	0.396	28
GA13 = Random, Tournament, SPX, Random, Fixed	0.0	20.1	1.49	4.80	0.91	100
GA14 = Random, Tournament, SPX, Random, Fitness	0.0803	20.9	5.76	6.82	0.144	18
GA15 = Random, Tournament, SPX, Gaussian, Fixed	$4.23*10^{-6}$	17.7	1.21	4.43	0.74	100
GA16 = Random, Tournament, SPX, Gaussian, Fitness	0.00559	20.2	3.41	6.35	0.052	5
GA17 = Grid, RWS, Two Point, Random, Fixed	0.00976	19.7	1.47	2.51	2.38	100
GA18 = Grid, RWS, Two Point, Random, Fitness	0.059	20.0	0.665	2.83	0.488	15
GA19 = Grid, RWS, Two Point, Gaussian, Fixed	0.022	18.04	0.814	3.58	1.58	100
GA20 = Grid, RWS, Two Point, Gaussian, Fitness	0.012	43.8	0.690	2.72	0.442	13
GA21 = Grid, RWS, SPX, Random, Fixed	0.00263	20.05	0.748	3.35	1.71	100
GA22 = Grid, RWS, SPX, Random, Fitness	0.0110	39.88	3.81	6.43	0.071	4
GA23 = Grid, RWS, SPX, Gaussian, Fixed	$6.62*10^{-7}$	21.8	0.690	3.22	2.04	100

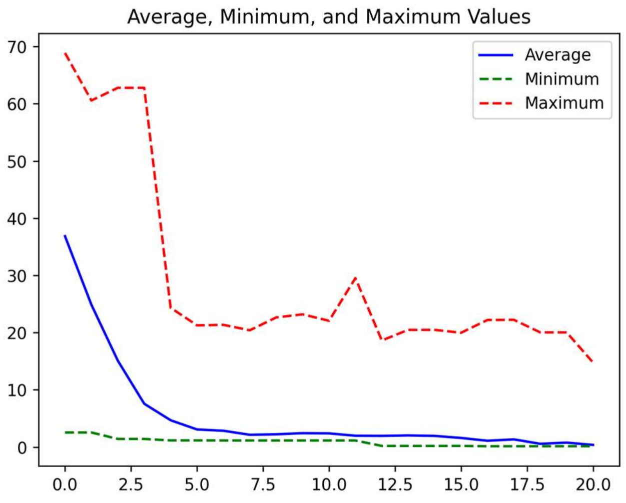
GA24 = Grid, RWS, SPX, Gaussian, Fitness	0.0703	19.8	0.885	2.95	0.327	18
GA25 = Grid, Tournament, Two Point, Random, Fixed	0.0648	22.1	0.480	2.47	0.620	100
GA26 = Grid, Tournament, Two Point, Random, Fitness	0.0202	15.1	1.49	3.21	0.254	14
GA27 = Grid, Tournament, Two Point, Gaussian, Fixed	0.0048	42.4	0.581	2.70	0.661	100
GA28 = Grid, Tournament, Two Point, Gaussian, Fitness	0.0761	22.0	1.99	3.23	0.354	37
GA29 = Grid, Tournament, SPX, Random, Fixed	0.0	19.9	0.873	3.28	1.00	100
GA30 = Grid, Tournament, SPX, Random, Fitness	0.00360	20.1	1.700	4.32	0.040	5
GA31 = Grid, Tournament, SPX, Gaussian, Fixed	0.0	35.5	1.28	5.12	1.03	100
GA32 = Grid, Tournament, SPX, Gaussian, Fitness	0.016	34.4	14.6	8.81	0.063	4

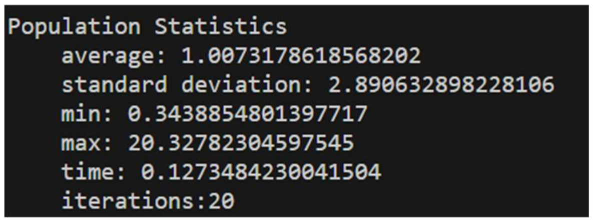
Table 2 Comparison of genetic algorithms using the Rastrigin Function

From this table a few things are made apparent. One is that the populations are converging well before 100 iterations. This can be seen as all the fitness terminations happen well before 100. Later, this paper will discuss when the fitness terminations are ending for the different complexity levels. Another item to be discussed later that is apparent from the table is the early convergence of SPX sometimes only taking 2 generations in the fitness termination, while having a low average and standard deviation. Also since SPX is a more complex crossover technique it takes longer for it to run.

2.2 CHANGE IN MUTATION TYPE, RATE, AND STRENGTH

For the previous example the mutation rate stayed the same, with the only change being between uniform random and gaussian. In this section, it is explored what happens when the rates are changed, and the strengths are changed are also changed. The number of generations has also been reduced to 20 and the population has stayed the same at 144. Below is a figure of GA1's (uniform random mutation) average, min and max for the Rastragin function with a length of 1 and a rate of 0.03:





```
Population Statistics
    average: 1.0073178618568202
    standard deviation: 2.890632898228106
    min: 0.3438854801397717
    max: 20.32782304597545
    time: 0.1273484230041504
    iterations:20
```

Figure 9 GA1's (uniform random mutation) average, min and max as well as population statistics for the Rastragin function with a length of 1 and a rate of 0.03

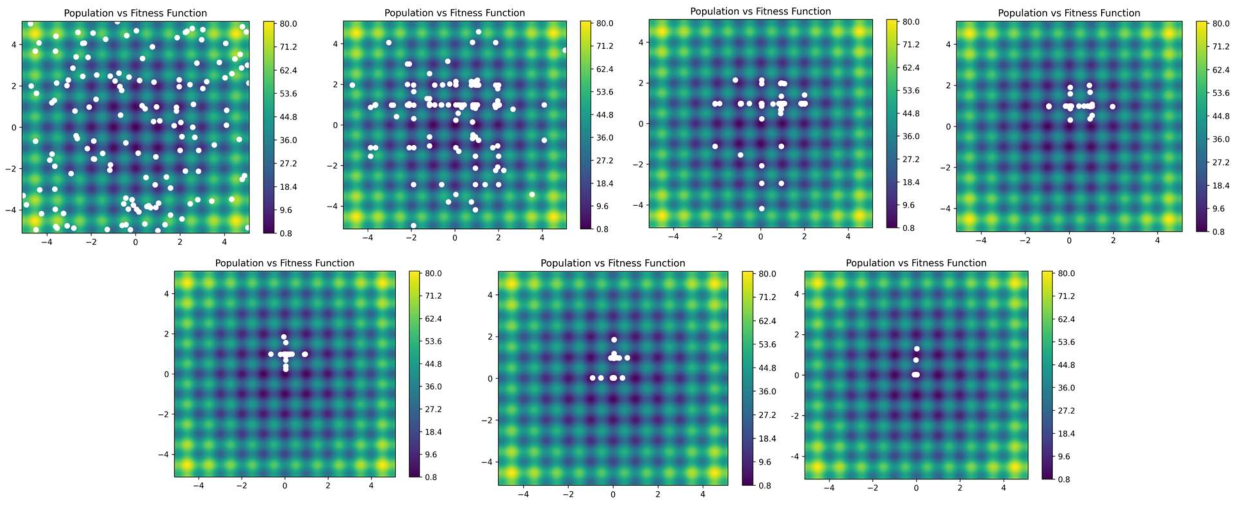
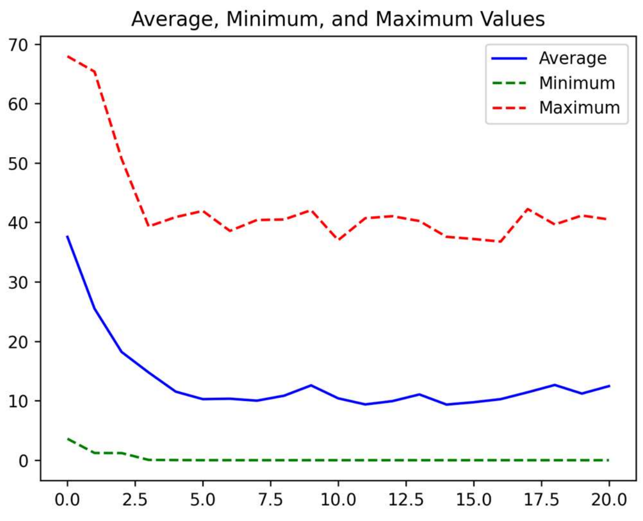


Figure 10 The population changes throughout the generations of GA1 with a length of 1 and a rate of 0.03. From left to right and top to bottom 0,2,4,6,10,15,20.

The population starts off randomly spread out through the search space. Then it starts to converge to the local minimums. Finally in the 15th generation it finds the global minimum through mutation. The mutation can still be seen in the 20th generation as there are chromosomes searching away from the global minimum, but only with a maximum of the length away.

2.2.1 Change in Mutation Rate

As the mutation rate increases the search spaces become more viewed around the current chromosomes. It does not expand the overall search space though since the selection process mostly eliminates fewer fit individuals from repopulating. A chromosome will most likely be eliminated before it can reproduce and the chances of that chromosome reproducing, and its offspring mutated, and reproducing are minimal. The average population does increase as does the standard deviation. Below is an example with the mutation rate set to 0.5. Another interesting point is that most of the mutations only occur in one direction, either x or y. Whenever mutation is raised, it does become more likely that a mutation will happen in both parameters of the chromosome. Also, the average is further away from the minimum when convergence happens.



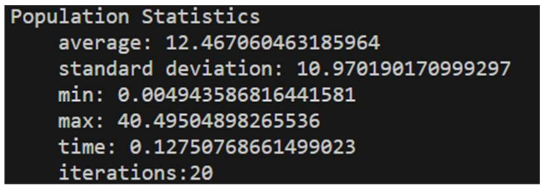

```
Population Statistics
    average: 12.467060463185964
    standard deviation: 10.970190170999297
    min: 0.004943586816441581
    max: 40.49504898265536
    time: 0.12750768661499023
    iterations:20
```

Figure 11 GA1's (uniform random mutation) average, min and max as well as population statistics for the Rastragin function with a length of 1 and a rate of 0.5

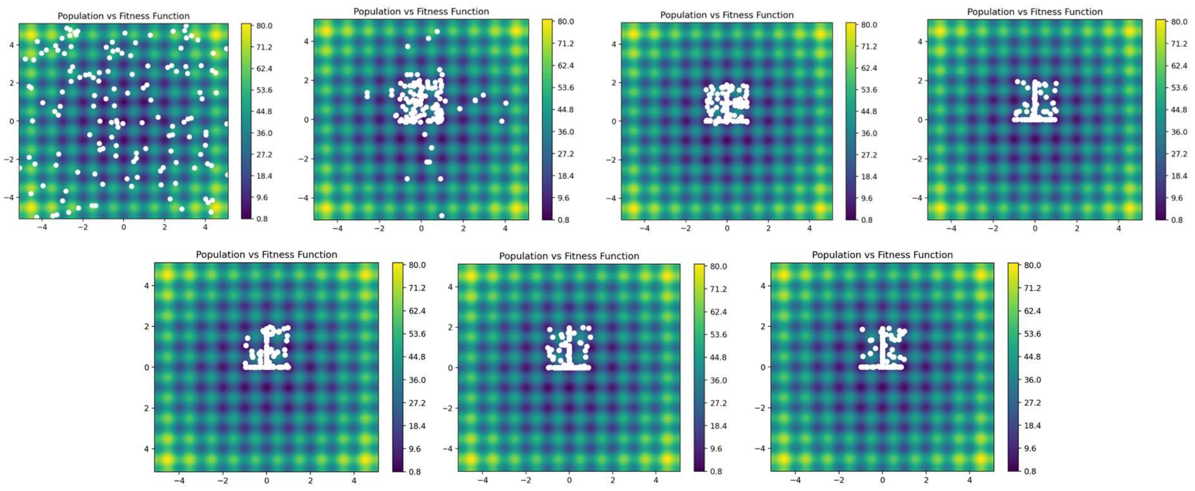
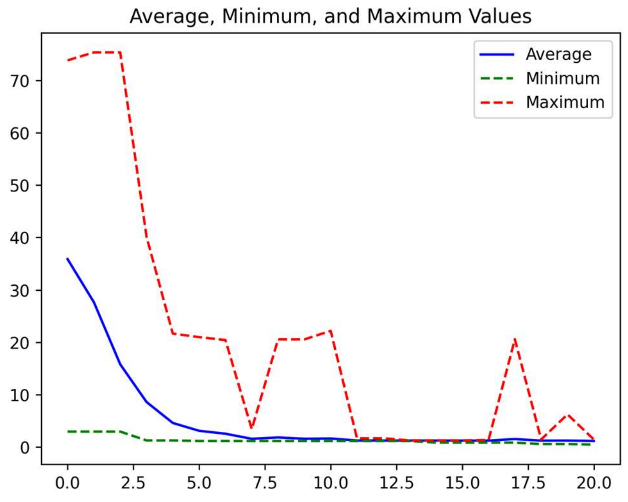
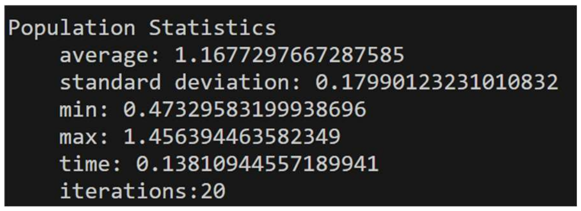


Figure 12 The population changes throughout the generations of GA1 with a length of 1 and a rate of 0.5. From left to right and top to bottom 0,2,4,6,10,15,20.

As the mutation rate decreases the exploration of the local areas decreases. The average is very close to the minimum. The maximum also stays around the minimum showing a very small search space. A small mutation rate also runs the risk of never finding the global minimum. For example, in the Rastragin if a search converges on [1,1], a local minimum, there is the risk of a mutation never reaching [0,0] if the program is not ran long enough.





```
Population Statistics
    average: 1.1677297667287585
    standard deviation: 0.17990123231010832
    min: 0.47329583199938696
    max: 1.456394463582349
    time: 0.13810944557189941
    iterations:20
```

Figure 13 GA1's (uniform random mutation) average, min and max as well as population statistics for the Rastragin function with a length of 1 and a rate of 0.005

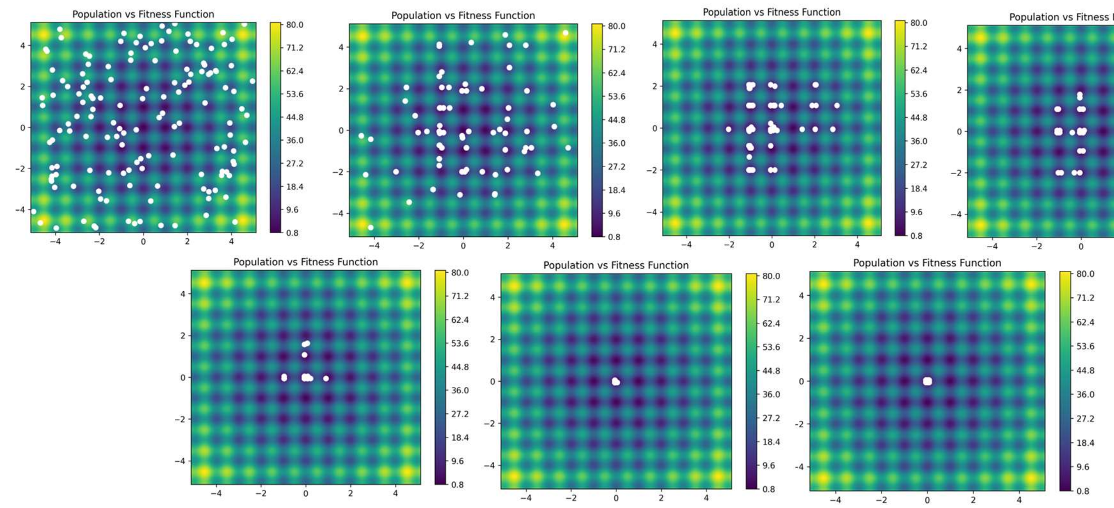
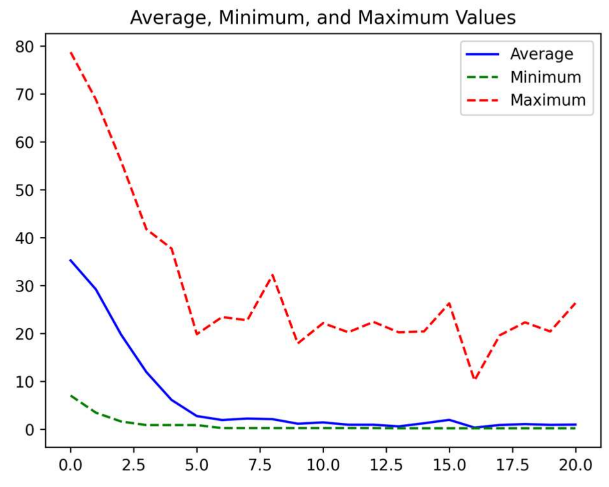
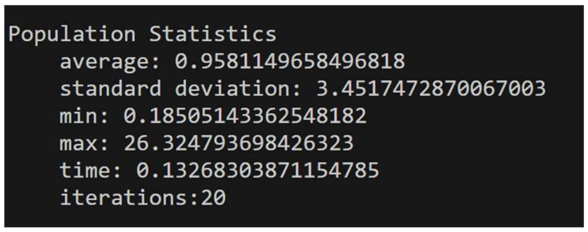


Figure 14 The population changes throughout the generations of GA1 with a length of 1 and a rate of 0.005. From left to right and top to bottom 0,2,4,6,10,15,20.

2.2.2 Change in Mutation Strength

The change in mutation strength will be represented by changing the length variable, (i.e. how far away can the mutation occur). With increasing the mutation strength, the overall population statistics don't seem to change much. The main change here is that the local search space is expanded, but since the population has converged to the global minimum, those mutated are normally not selected for reproduction. With this, it is less likely to find the local minimum since it is searching more space.





```
Population Statistics
    average: 0.9581149658496818
    standard deviation: 3.4517472870067003
    min: 0.18505143362548182
    max: 26.324793698426323
    time: 0.13268303871154785
    iterations:20
```

Figure 15 GA1's (uniform random mutation) average, min and max as well as population statistics for the Rastragin function with a length of 2 and a rate of 0.03

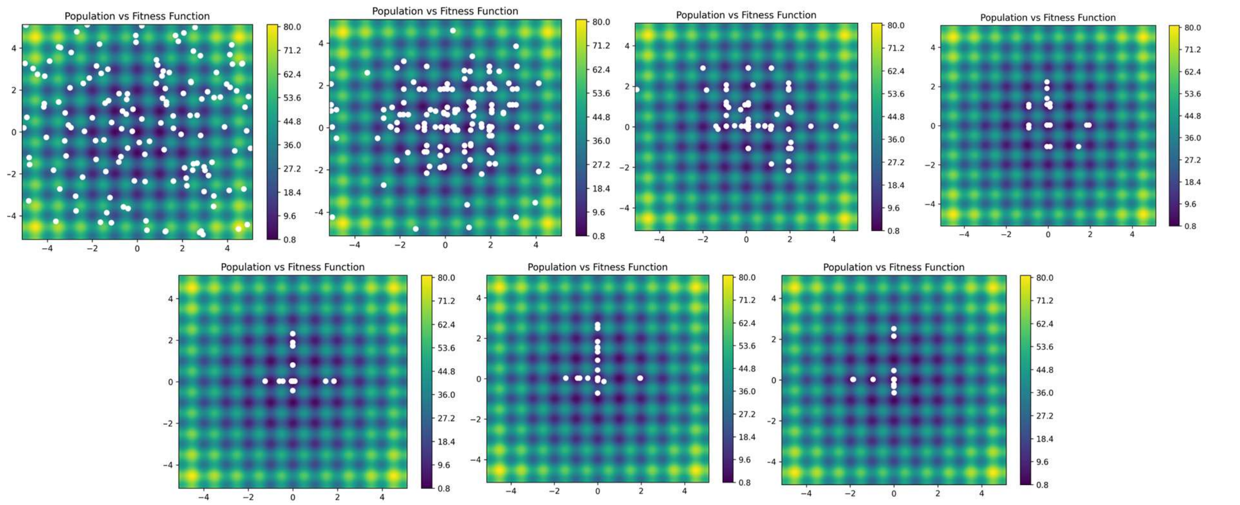
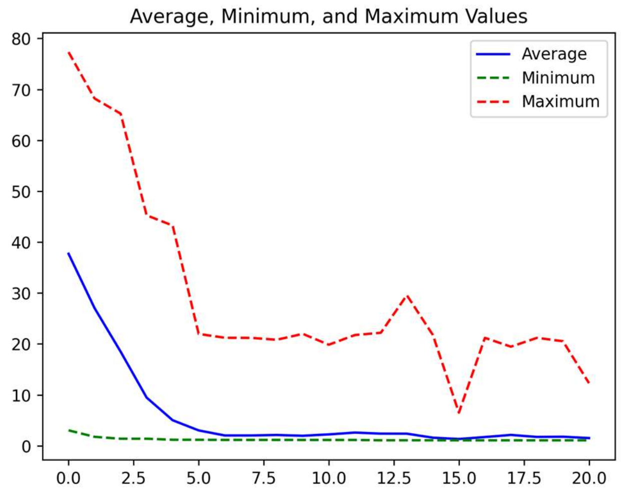
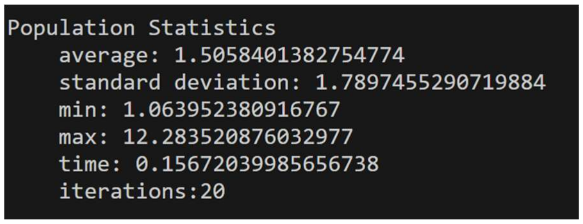


Figure 16 The population changes throughout the generations of GA1 with a length of 2 and a rate of 0.03. From left to right and top to bottom 0,2,4,6,10,15,20.

With decreasing mutation strength, the risk for not searching enough space is created and getting stuck in a local minimum. This is because the local search space has decreased. Here you can get the a better chance at the local minimum since you are searching less space.





```
Population Statistics
    average: 1.5058401382754774
    standard deviation: 1.7897455290719884
    min: 1.063952380916767
    max: 12.283520876032977
    time: 0.15672039985656738
    iterations:20
```

Figure 17 GA1's (uniform random mutation) average, min and max as well as population statistics for the Rastragin function with a length of 0.5 and a rate of 0.03

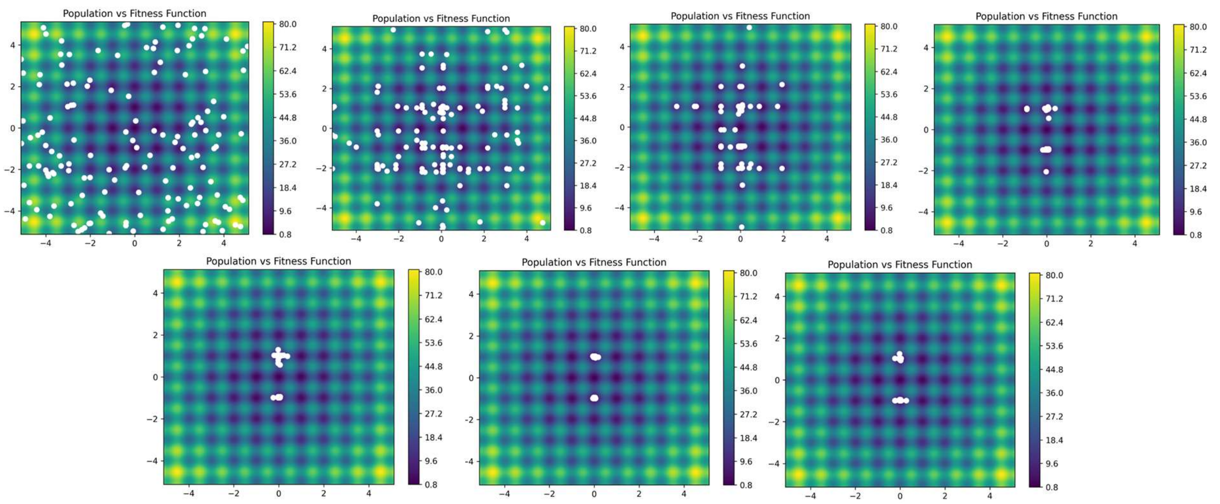


Figure 18 The population changes throughout the generations of GA1 with a length of 0.5 and a rate of 0.03. From left to right and top to bottom 0,2,4,6,10,15,20.

2.3 COMPARISON OF DIFFERENT COMPLEXITIES

The different complexities of fitness functions illustrate how genetic algorithms perform with increasing complexity. GA1 and GA2 were chosen and tested with each fitness function. There were 20 generations for GA1 and a population size of 144. The mutation rate was 0.03 with mutation length of 1. They were tested in a two-dimensional space. As the problems became more and more complex the average starts to increase as does the standard deviation. Also, when looking at the fitness termination the number of generations needed increases with complexity as well. Due to the Rosenbrock having such a large search space with values significantly increasing as you move further away from the minimum it appears to converge earlier, than it does as it narrows the space down, but doesn't find the optimum point right away.

Fitness Function	**Minimum Value**	**Maximum Value**	**Average Value**	**Standard Deviation**
Sphere	0.00000904	2.62	0.0363	0.281
Rastragin	0.489	20.5	0.831	2.16
Rosenbrock	0.0352	166.52	2.61	15.5

Table 3 Performance vs Complexity of GA1

Fitness Function	**Generations for Fitness Termination**
Sphere	5
Rastragin	14
Rosenbrock	496

Table 4 Generations vs Complexity of GA2

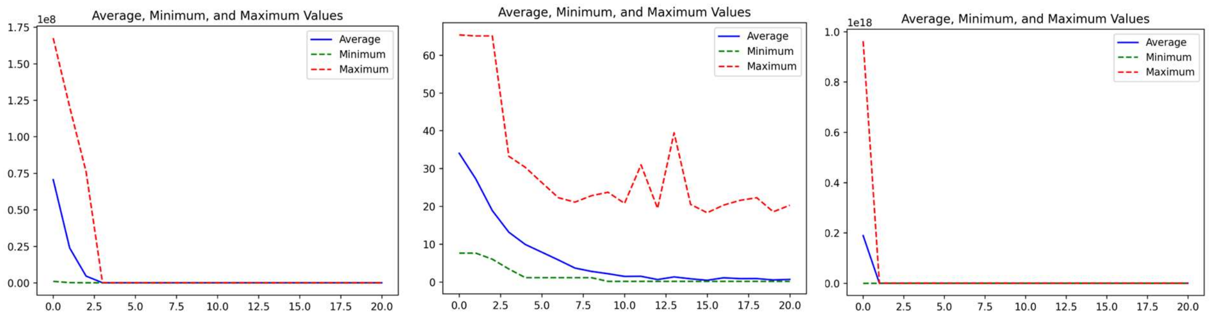
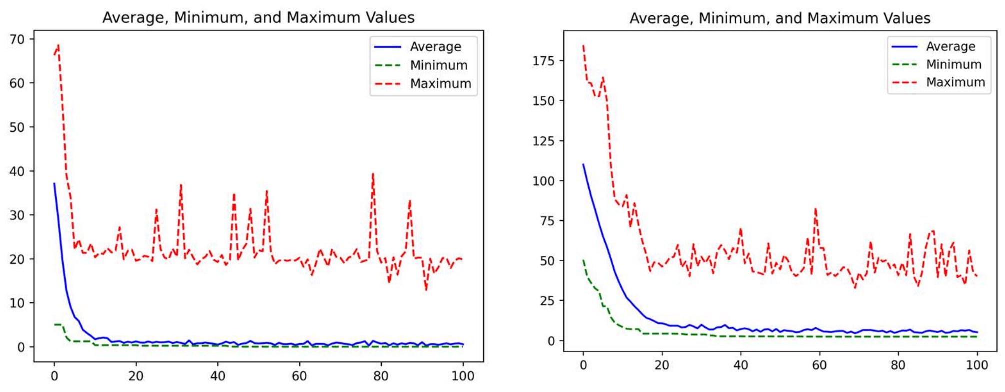


Figure 19 Population Statistics Overtime. From left to right Sphere, Rastragin, Rosenbrock

2.4 Changes in Dimensions

As the number of dimensions increases so does the complexity and there for the time it takes for the population to converge. GA1 was chosen and tested with the Rastragin function. There were 100 generations with a population size of 144. The mutation rate was 0.03 with mutation length of 1. In the experiment the dimensions are increased looked at from 2, 6, 10, 15 on the Rastrigin function to see the changes in time of convergence. With 2 dimensions the population converges very quickly at around 10 generations. Whenever the dimensions are increased to 6 the population converges closer to 25. With 10 dimensions this number grows to 60 and with 15 it seems to have not converged at 100.



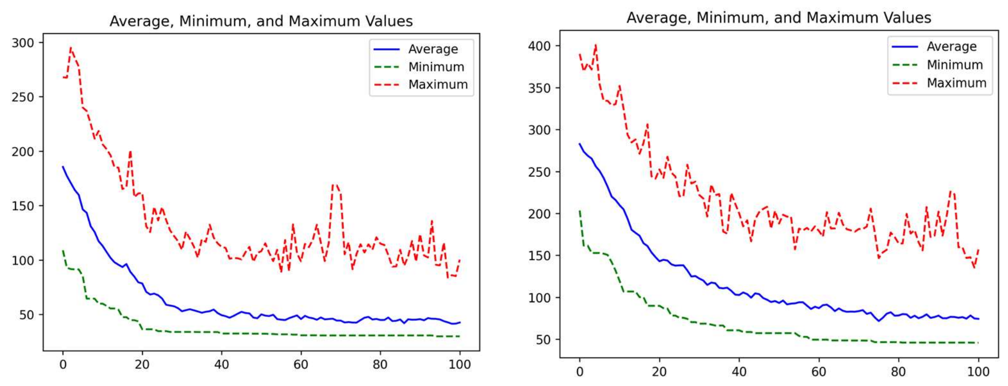


Figure 20   Average, Minimum, and Max Graphs for varying dimensions

Even though the populations appear to have converged on the graphs it does not mean that the optimal solution was found. For the dimensions above 2, the optimal solution had a few genes still at 1. Where the optimal solution for the Rastragin is [0,0,…,0]. To find the optimal solution GA1 will have to run for longer and get a mutation to lower the point to 0.

A valuable conclusion made here is that running just one genetic algorithm can get stuck on a local minimum as the problem gets more and more complex. Thus, it is important to run a genetic algorithm multiple times to better improve the chance of finding the optimal result.

2.5 SPX vs Two Point Crossover

The next experiment was to see the differences between Simplex Crossover (SPX) and Two Point Crossover. GA1 and GA5 were chosen and tested with the Rastragin function. There were 100 generations with a population size of 144. The mutation rate was 0.03 with mutation length of 1. They were tested in a two-dimensional space. One of the first things noticed is in SPX the population all converges in a circle in unison. The two-point crossover while converging has more searching outside of the population mass.

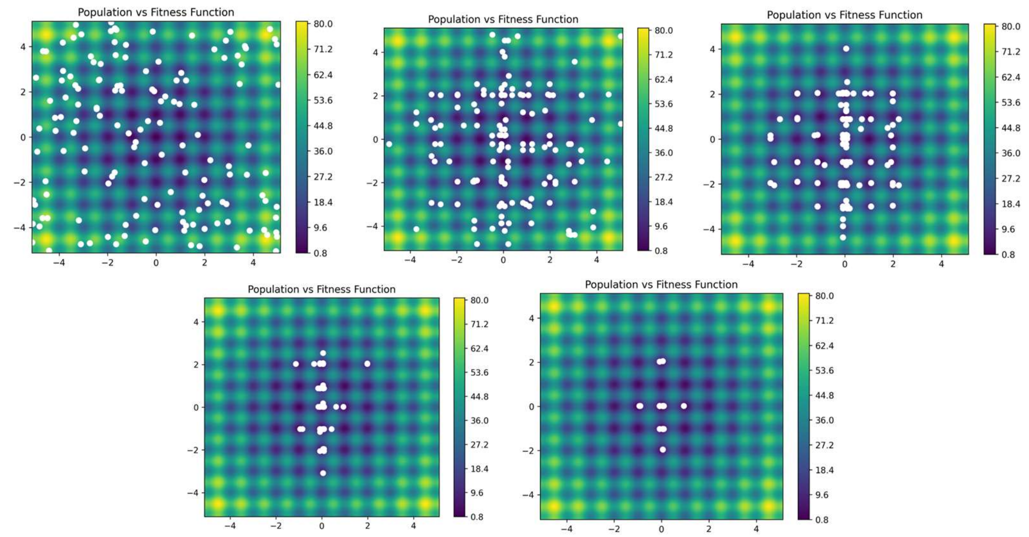


Figure 21 The convergence of GA1 using two-point crossover. From left to right, top to bottom generations: 0,2,4,6,8

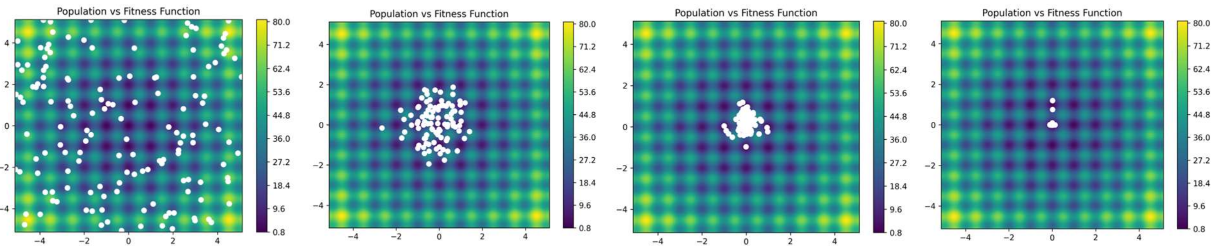


Figure 22 The convergence of GA5 using SPX crossover. From left to right, top to bottom generations: 0,2,4,6

Another item of importance is the fact that the SPX crossover converges faster than the two-point crossover. It only took the SPX 6 generations where the two-point was on the verge of converging on the 8th generation. The two-point also performed better in a multi-dimensional space which was surprising. The experiment was ran, due to the belief that the SPX would help solve problems with more dimension. Below are the results of the SPX and the two-point crossover in a 5-D space. Each one ran 10 times. The values are average(standard deviation) – minimum.

SPX	Two-Point
7.22(12.2) – 0.744	5.84(6.72) – 3.41
6.68(11.9) – 1.42	4.08(5.77) – 2.51
33.4(14.7) – 5.73	4.66(6.45) – 2.05
34.1(12.8) – 8.71	2.44(6.35) – 0.285
5.99(9.46) – 1.45	7.63(8.19) – 4.36
9.17(9.29) – 3.37	4.68(7.78) – 1.93
6.50(9.04) – 2.16	4.70(6.91) – 2.35
23.4(14.15) – 3.63	6.35(7.59) – 3.29
19.9(13.4) – 5.14	6.76(7.02) – 4.18
11.9(8.89) – 4.15	6.27(6.21) – 3.92

Table 5 Comparison of SPX and Two-Point. Average(Standard Deviation) - Minimum

3 Summary and Future Work

In conclusion, an evolutionary algorithm is an optimization program inspired by evolution with many different implementations to optimize the program itself. These optimizations hope to improve the chances of finding the most fit individual or otherwise known as the best solution to the problem space. The evolutionary algorithm is implemented by initializing a population with a group of chromosomes. Every chromosome has n-dimensions of genes or properties. These genes determine how fit an individual is.

The population goes through a series of generations to discover the optimal result. Each generation first has its fitness evaluated. Then the population goes through a selection process to determine who will be the next set of parents based on their individual fitness. After the parents are selected, crossover occurs where the next generation is produced. Finally, mutations occur at a slim rate. Then the next generation starts. This continues until the termination criteria is complete. There are many ways to do each of these tasks. For the implementation in this paper initialization was done using random and grid. The fitness functions that were evaluated were the Sphere, Rastragin, and Rosenbrock functions. The selection was done through roulette wheel selection and tournament selection. Elitism was also performed. Crossover was accomplished by two point and simplex crossover. Mutation was performed through gaussian mutation and uniform mutation. Termination was determined by a fixed number of generations or by a minimum fitness being met.

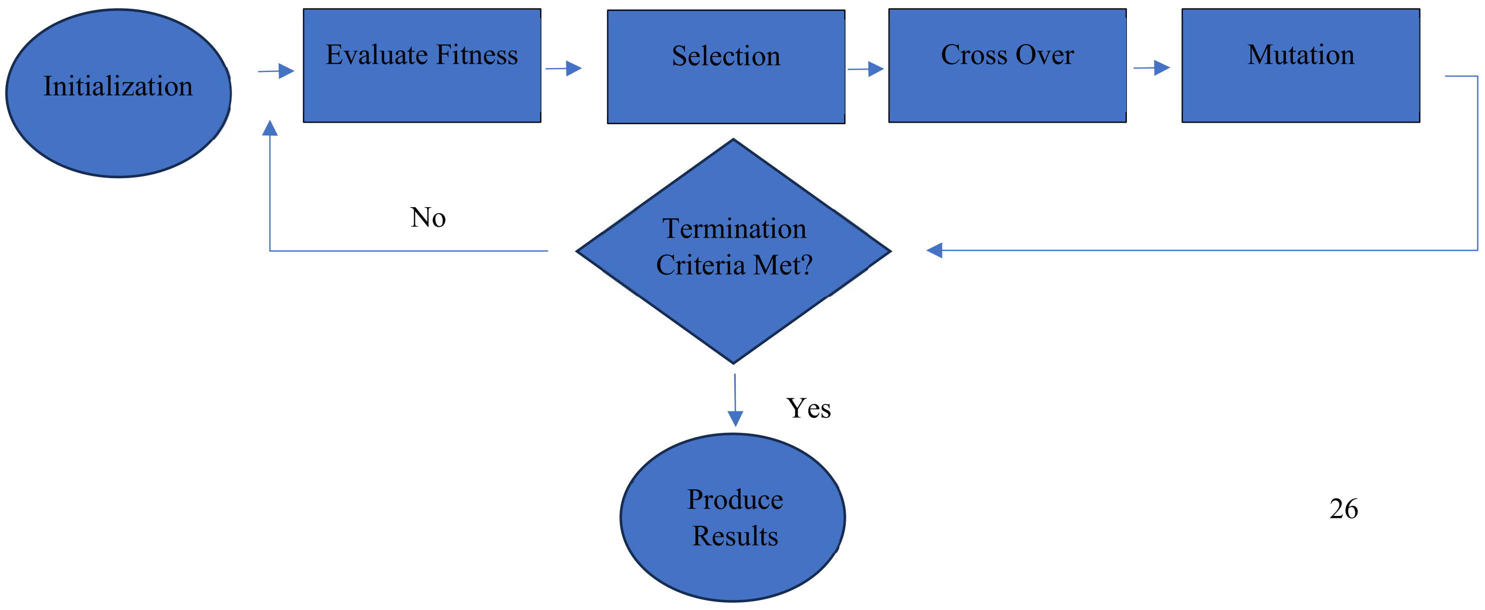


Figure 23 Illustration of a genetic algorithm

Many important conclusions were drawn throughout this paper. The first important conclusion was through the comparison of termination methods found from the high-level overview. The fixed generation termination typically found the best solution and made sure the population converged. Although often the population converged well before the genetic algorithm stopped. The fitness limit termination always found a solution less than the threshold, but this wasn't always the optimal solution. Also, often the population did not have enough time to converge.

Future work based off this conclusion is the ability to end the program on convergence. This way the program is not running for excess time, but as well always converges.

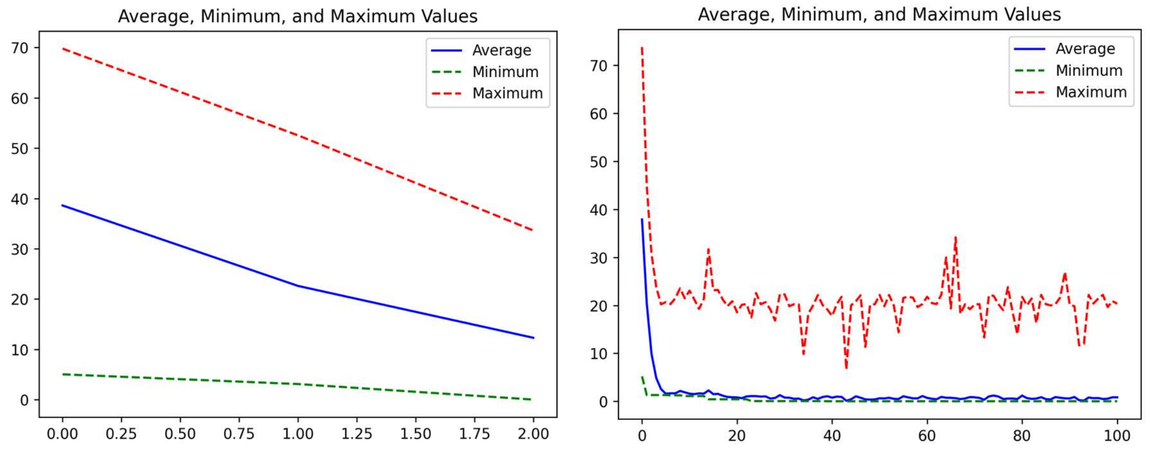


Figure 24 The image on the left is a population ending before converging and the image on the right is a population ending well after converging.

The second conclusion made was the effects of mutation strength and rate. As the mutation rate increases the search spaces become more viewed around the current set of chromosomes. It does not expand the overall search space though since the selection process mostly eliminates less fit individuals from repopulating. As the mutation rate decreases the exploration of the local areas decreases. The average becomes very close to the minimum. The maximum also stays around the minimum showing a very small search space. A small mutation rate also runs the risk of never finding the global minimum. With increasing the mutation strength, the overall population statistics don't seem to change much. The main change here is that the local search space is expanded, but since the population has converged to the global minimum, those mutated are normally not selected for reproduction. With decreasing mutation strength, the risk for not searching enough space is created and getting stuck in a local minimum. Here you can get a better chance at the local minimum since the program is searching less space.

Future work for this project based on this conclusion will be having a variable mutation strength and rate. This will achieve the benefits of both. The mutation strength will be high in the beginning with the rate lower. In the beginning, a larger search space is wanted to try and find the global minimum area. Then once the population has converged the strength will decrease and the mutation rate increases. This will allow a better search for the local minimum. In terms of the Rastragin, while the population has not centered around [0,0] it will be optimized for global search. Once the population has converged around [0,0] it will be optimized for local search. It is important to keep a small amount of the mutation with a higher strength and a lower mutation rate, in case the program is stuck in a local minimum.

Another valuable conclusion is that running just one genetic algorithm can get stuck on a local minimum as the problem gets more and more complex. Thus, it is important to run a genetic algorithm multiple times to better improve the chance of finding the optimal result. Another solution is setting up an island algorithm.

Future work for this project will be the implementation of an island algorithm. An island algorithm has multiple genetic algorithms that can have different implementations. These algorithms run separately until a certain criterion is met. This is often a certain number of generations have passed. Then each algorithm will share some of its current chromosomes with the other islands.

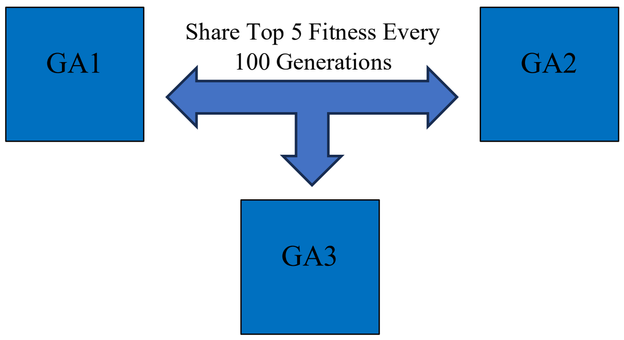


Figure 25 Illustration of Island Genetic Algorithm

Nevertheless, through this paper many important conclusions were drawn about genetic algorithms as well as a further understanding of how they work. Ways for genetic algorithms to be implemented outside of the fitness functions through parameters for neural networks or regression functions.