Computational Intelligence 2020 **(Due: 30/04/2020)**

Submission Assignment #3

Instructor: Jakub Tomczak *Name:* Tyron Landman, *Student id:* 2674357

1 Introduction

In this report, I will be analysing an evolutionary algorithm to determine how the different components of the algorithm affect the overall performance. An evolutionary algorithm (EA) is an algorithm that tries to imitate real-world natural selection through a process similar to survival of the fittest. EA's are useful when we have a function and can't calculate gradients (like the one I will be using in the testing phase of the report).
This is done by generating a sample population that is then run through a fitness (also known as objective) function to determine each element in the populations fitness. With these values, we are able to remove the worst ones. Once this has occurred, a recombination function is used to create new candidate points, similar to how a couple may have children. As the new candidate points are created, they can then be run through a mutation function. This function slightly alters the values for each element in the population to provide a bit more variability. Once this is complete, the process once again goes through the fitness analysis, and repeats itself over a given number of iterations.
The components in particular we will be focusing on in this report will be the Mutation, and the Recombination function. I will run a few tests with multiple population sizes, as well as over a different amount of generations including both the Mutation and Recombination function. Some tests will also be without to determine how each particular function affects the algorithm.

2 Problem statement

As with most optimizing algorithms, the evolutionary algorithm have a few potential drawbacks we will take into account and try to work around through the implementation and experimental phases of this report. In regards to the fitness function itself for instance, we may end up favoring particular individuals which may lead us a crowding effect where the population starts to stay within a local minimum, resulting in a dense population. Alternatively, there is the problem of vanishing selective pressure, where the fitness scores between individuals is too small as well.
As I will be using a survival fitness function, these two issues become more likely as it focuses more on the candidates that do well. So to assist in avoiding this issue, when we make child/candidate points we will randomise between the parents to help spread the distribution more evenly. Further to this, when using the mutation function it will further decrease the chances of these two events occurring as both recombination and mutation allows for a small probability to escape any local minimums or plateaus.

What will be difficult with this algorithm, and potentially unanswerable in this report is whether the algorithm used is optimum itself. As there are multiple variations on how to select new candidates, multiple variations of recombination functions, and many different ways of mutating those candidate points, we will restrict ourselves in this report to just testing the impact the mutation and recombination functions have. This will help us avoid further issues of optimization by fine tweaking each function itself for a better overall performance.

3 Methodology

To address our problem statement, as mentioned above, we will be using a mixture of the components "Mutation", and "Recombination". What I have implemented is a class for this process that will allow me to decide whether or not Recombination or Mutation is used when generating new candidates. This will allow us to see the performance metric of how the EA will work without Recombination and Mutation, and with Recombination and Mutation. It will also show us how the EA functions using only one of the two.
For the Recombination method, I have chosen to use random selection on two "parent" points. It will take these parent points and average the individual values associated to them respectively and create a child/candidate. This process will repeat itself until an entirely new population of equal size of the original is created whereby it will then be mutated and run through the fitness survival selection function.
For the mutation function, a simple function was created that adds a small integer value to the child candidate population, ensuring it does not break outside of our given domain.
For our experiment, each "persons" in our population will have a vector value containing four variables labeled:
a_0
n
$beta$
$alpha$

For each variable, we will also add a domain. The domain we will use will allow us to restrict the search space to a manageable amount. This domain will be as follows:
$a_0 \in$ [-2,10]
$n \in$ [0,10]
$beta \in$ [-5,20]
$alpha \in$ [500,2500]

Once implemented, I will be able to see any correlation/convergence of optimized values for each individual variables pertaining to each persons as the EA is run. I will also include a timer to see how each component function affects overall efficiency as well.
Lastly, I will plot the end performance metric of each runs fitness score to see how the adding/removal of functions affects the performance.

4 Experiments & Results

For the experiment to begin, some values for the population size, generation count, and standard deviation were required. Due to the mutation function, the standard deviation had to be below 1. As a result, for the initial testing phase the default values chosen for the three hyper-parameters are:
Population size: 20
Generations: 10
Standard Deviation: 0.4

To determine the performance of the EA with the different combination of components, I will be altering the population size.
To begin, we will start the testing with five different, but increasing population sizes on the EA with no recombination or mutation:

Population	Time taken	highest value	lowest
10	6.04	119.69	115.07
25	15.82	101	39.21
50	29.11	212.2	39.21
100	63.41	116.41	39.21
500	363.48	115.63	39.21

Table 1: EA, No Recombination, no Mutation

From the above table, an immediate point that stands out is the population being at 50 having the highest fitness value by a large amount. We can tell from this table as well that with no recombination or mutation function the EA converges very quickly to having a mean fitness score per generation at 39.21. This is not a good sign but as mentioned before was an expected result of the fitness function that would potentially be solved by introducing the recombination and mutation.

Population	Time taken	highest value	lowest
10	6.19	119.54	114.14
25	15.95	109.35	93.64
50	28.63	158.41	93.93
100	63.98	116.51	97.51
500	348.06	115.84	97.84

Table 2: EA, No Recombination, Mutation

For the above table, mutation was added to alter the values slightly per generation. As we can see the lowest value doesn't drop nearly as low as in the previous tests but comes to rest closer to around the 97.84 mark. What is also interesting to note is that the population size at 50 still yields the highest fitness value as well. Further to this, although an extra step was added to the EA, the time taken to complete the calculations wasn't much different that not having the mutation component at all.

Population	Time taken	highest value	lowest
10	5.29	118.56	115.16
25	15.93	114.54	109.2
50	25.24	117.32	104.12
100	58.4	116.65	102.52
500	305.15	115.52	98.58

Table 3: EA, Recombination, No Mutation

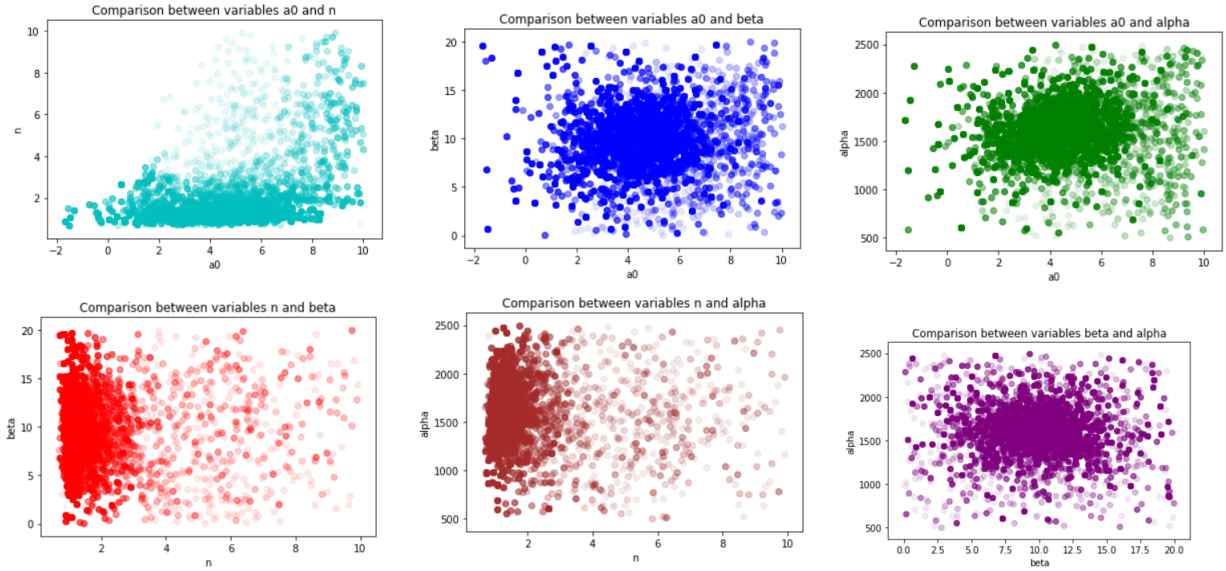
Thirdly, we tested how the EA would perform using the component Recombination with no mutation. As we can, there is a much slower decline in both the highest and lowest values. The lowest value halting at 98.58. Another note to make is that although an extra step was including, what is interesting is that the time taken is actually much faster than both previous tests. However it must be noted this may be a result of the hardware with external factors affecting this.

In our last test, I ran the EA using both recombination as well as mutation. This had some interesting results because irrespective of the population size, it seems that the highest and lowest values seemed to be converging. This is a great sign as it signals that the best fitness score (or the minimum) must be somewhere between these two scores.

Finally, using the information gained from the above tables we can see that we were able to ascertain via the previous tests I implemented another piece of code to display the convergence of the individual points.
This is useful as we can see that for each point, there is an optimal value that will be produced that gives us the optimized results for our fitness function.

Population	Time taken	highest value	lowest
10	4.02	118.55	114.65
25	14.38	114.14	96.38
50	26.48	117.05	97.88
100	56.91	116.23	97.62
500	309.95	115.55	98.26

Table 4: EA, Recombination, Mutation



From the above images, we can see that there is definitely a point where each point in relation to each other does in fact converge to find the optimal for that particular variable. As an example, from the two graph of $a0$ in relation to *alpha* and *beta* we can clearly see that the optimal points do begin to converge around the center. What is interesting to note however is from the graph of $a0$ and n. In this graph we can see that in the early generations the points become darker converging on the right side of the graph. Then it begins to move down and converge on the bottom half, fairly evenly spread between the zero and the 8.

5 Conclusion

In conclusion, evolutionary algorithm can be very useful when a function is presented where we can't calculate the gradient. From the algorithm, we were able to not only measure the overall performance of the individual parts of a "persons" within the population, we were able to slowly converge on the optimized values of individual parts themselves.
The downside to EA however is it is very easy to get a crowding of points and convergence on points that may not be accurate at all. This can be navigated with a well implemented recombination and mutation function as we saw but further testing with different mathematical approaches for these function may have altered the end result itself revealing if what we found was a local or global minimum.