2019

Hill climber and Genetic Algorithm

AIAB REPORT

168908

Contents

Genetic Algorithms

Keywords : Genetic algorithm, demes, maxima, fitness landscape

Abstract

In computer science, a genetic algorithm is a process of doing natural selection harnessing the computational power of the computer. Genetic algorithms are commonly used to create solutions to optimization and search problems by using the biologically inspired operators such as mutation, crossover and selection. It was introduced in the 1960s by Holland, J. H. (1962) based on the concept of Darwin's theory of evolution. In this research study, it aims to implement hill climbers and microbial genetic algorithms to solve the knapsack problem and then observe the effect on their performances when a number of crossover and generation values are changed. And compare the different algorithms' performances.

Hypothesis: The greater population of hill climbers should give better results than a smaller population of hill climbers. The hill climbing algorithm will have worse performance than the microbial genetic algorithm. Hill climbing will experience local maxima so microbial should be able to find the most optimal solution easier and with lesser generations. Also, I expect crossover value to influence the results, with higher crossover having better performances.

1 Introduction

In the 1950s, Alan Turing proposed the concept of a learning machine which uses the concept of evolution. By the early 1960s, books by Fraser and Burnell(1970) described the methods that early biologists used to achieve computer simulation of evolution. Fraser's simulations already had included all the essential elements of modern genetic algorithms. Rechenberg(1965) introduced an optimization method called "evolution strategies", this was used on many devices and one of the most notable was the airfoils. The main idea is to have a population of individuals and allowing them to mutate slightly each iteration (simulating each generation) and if the mutated individuals are better than the parent then the mutated version replaces the previous version otherwise if the mutated version has a worse fitness than the parent then nothing will be changed. The idea is that during each iteration, the individuals will improve slightly and with a greater population, eventually the individual with the highest fitness is the most optimal solution. Hence, it is easy to see how genetic algorithms are an idealized computational model of Darwin's theory of evolution to solve real-life problems.

The overall concept of how genetic algorithms help to find the optimal solution is as follows described by Yu, X. and Gen, M. (2010), each individual has its own genes which are represented in the computer by binary values of 1 or 0. The value of 0 means not taking the corresponding benefit and the number 1 being taking the corresponding benefit. And the resulting phenotype which is the actual characteristics determined from its genotypes would show exactly how well it solves the problem. It can be observed how the optimal solution to the problem can be solved via this method. In my research, I have used the hill climbing genetic algorithm and the microbial genetic algorithm to examine the effects of mutation rate, generations and crossover values on the two types of algorithms and how their performances differ.

1.1 Knapsack problem

The knapsack problem to be solved is about the best possible solution must be found. The bag contains a set of benefit and

corresponding volume associated with them. There is the constraint that the maximum volume (figure 2) must not be reached otherwise the overall fitness must be set to zero. The fitness value is to be obtained by adding together the total benefits of the chosen items (figure 1), the optimal solution should output the highest fitness value possible without going over the constraint.

The goal is to maximize value,

$$\Sigma_i^N B_i$$

Figure 1 the sum of all the benefit values

subject to the constraint,

$$\Sigma_i^N V_i \leq V.$$

Figure 2 The total volume must not exceed the limit

The assumption in this problem is that each item can only be picked once so the corresponding value in the genotype must be zero if the item is not taken or set to be 1 if the item is taken. There are 10 items that can be taken so a 10 binary numbered genotype is set up to represent the idea which items are taken or not.

The given problem is shown in figure 3, it is representing the weight(volume) and the corresponding benefit values, each column is represented by a value in the genotype.

Item	a	b	c	d	e	f	g	h	i	j
B	5	6	1	9	2	8	4	3	7	10
V	3	2	4	5	8	9	10	1	6	7

Figure 3 – 10 items in the table labelled alphabetically with corresponding benefit and volume

1.2 Hill Climbing Genetic Algorithm

The first algorithm of the study is the most basic genetic algorithm called hill climbing. The idea of hill climbing genetic algorithm is to find a sufficiently good solution to the problem which is called a heuristic function because it does not guarantee the best solution to the problem however, it is very simple to implement and can be good to a solution to the problem. The name hill climbing comes from the approach that starting with a sub-optimal solution is like starting at the bottom of the hill, and as the solution is optimized gradually, it is almost like hill climbing in the real world and eventually reaching the top of the hill which is the equivalent best solution. However, often the solution is stuck in what is known as a local maxima, in which the solution seems to be on the top of the hill but there could well be a better solution to the problem and this could be visualized in such a way that the climber began at the bottom of the alps and reached the top which is Mont-Blanc and it may seem to be the best solution but there may be a globally optimal solution which is higher like Mount Everest but because of the hill climber being unable to see the other optimal maxima, that particular hill climber will be stuck on the local maxima. There are three techniques to be respected: firstly, the algorithm must construct a sub-optimal solution which obeys the constraint of the problem, for this knapsack problem it would be the capacity in terms of volume. Secondly, the solution must be improved step by step which is represented by the generations in hill climbers. Thirdly, the solution must be improved until no more improvements could be made, this can be

represented by the flat lines on the fitness vs generation graph.

In my study, a random genotype will be generated, and it will have a random fitness value which can be a point in the graph (figure 4). The algorithm will loop around mutating the current genotype until it can find a genotype with better fitness, so the idea is that the current hill climber position should never go downhill. The idea that the hillclimber may be stuck on the local maxima labelled on figure 4 means the hill climber will not be able to climb to the global maxima.

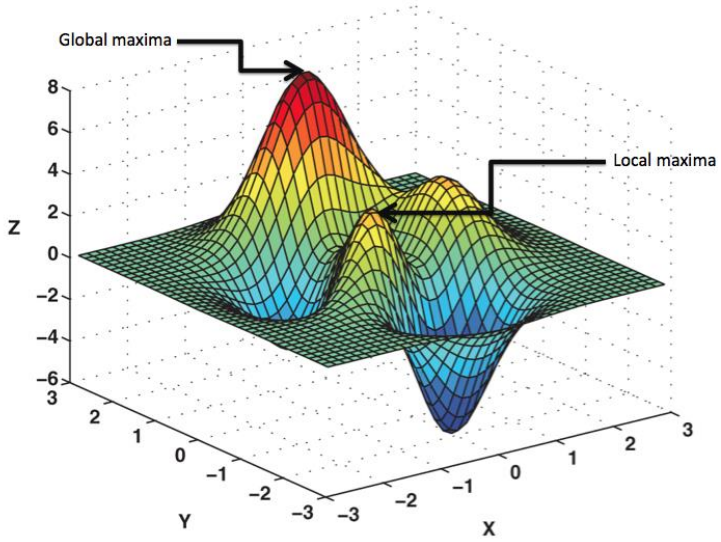

Figure 4- illustrating an example of fitness landscape showing global maxima and local maxima. In a fitness landscape, with randomly generated genotypes, it can be on any part of this landscape and the objective is to reach the global maxima.

1.3 Microbial Genetic Algorithm

Ian Harvey(2001) from the University of Sussex came up with this microbial genetic algorithm. It is a stripped down and reduced version of the conventional algorithm that can be interpreted in terms of horizontal gene transfer like bacterial conjugation. The microbial genetic algorithm has proven to be similar in terms of performance as the conventional method but is much simpler to be programmed. As the name suggests, the obvious inspiration for this algorithm came from bacteria, binary fission and concepts from trivial geography.

This particular algorithm can be implemented by randomly selecting two individuals in the population and by using tournament selections a winner and a loser is chosen based on their fitness values. A crossover is done by copying some of the winner's genes to the loser and the loser is mutated again to be placed back into the population. The process is repeated until no improvement could be made. In the paper by Mitchell, M., Forrest, S., & Holland, J. H. (1992, December), it is mentioned: "that Genetic Algorithms will outperform hillclimbers because crossover allows the powerful combination of partial solutions."

2 Method

I used MATLAB to implement the following methods following example concepts from Yang, X. (2006) book on An introduction to computational engineering with Matlab.

Genotype: the size of the number of items, you have a 1 or 0 to take the item or not
Phenotype: benefit and volume
fitness: the sum the benefits unless the volume is over 0, set 0.

2.1 Single Hill Climbing Genetic Algorithms

Pseudo code for Single Hill-Climbing Genetic Algorithms

1. *Create a random genotype g1 and copy it to a temporary array g0 ← g1*
2. *Mutate g1*
3. *Evaluate new fitness of g1*
4. *If fitness(g1) > fitness(g0)*

   then g0 ← g1

   else

   do nothing

   end;
5. *For n generation, go to 2*

Benefit and volume are stored in two separate vectors which match the one shown in figure 3. I created two functions, one is to work out the fitness and the other being the mutation function.

The fitness function works by setting the maximum capacity as 20 and having an if-else statement for asking if the volume has exceeded the maximum capacity and according to that, a value will be set to either 0 if exceeded or other values if not exceeded.

The mutation function includes randomly selecting a number between 1 and 10 which is basically choosing a random point in the genotype. The genotype is first copied to another matrix called mutation and if the chosen point in the matrix is 0 then it must be flipped to 1 or vice versa. This ensures the chosen position in the genotype is mutated.

For the single hill climber, a for-loop goes through 1 to 100 generations and within the loop, the mutated gene compares fitness with the current genotype and if the fitness of mutated gene is better than it overwrites the current genotype. This happens for 100 generations and the fitness value will be plotted on fitness against generation graph to see how it improves over time.

2.2 Population of Hill Climber Algorithm

Like the single hill climbing genetic algorithm, this version adds another for-loop which iterates over the number of populations. This will be used to examine how the number of the population influences the finding of the optimal solution. Especially this should solve the problems faced by the single hill climber in that, once there is a population, it is hard to be stuck on a local maxima. It can be visualised like having several more hill climbs randomly distributed around the landscape and that increases the chance that at least one of those hill climbers would reach the global maxima.

1. *Initialise a matrix G, with a population P*
2. *For every generation:*
3. *For every G in population:*

4. *Create a random genotype g1 and copy it to a temporary array g0 ← g1*
5. *Mutate g1*
6. *Evaluate new fitness of g1*
7. *If fitness(g1) > fitness(g0)*

   then g0 ← g1

   else

   do nothing

   end;
8. *For p population, go to 3*
9. *For n generation, go to 2*

The number of populations will be changed from 10 to 50 to 100, to see the effect it has on the results. And if predicted correctly, with more populations, the chance of finding the global maxima is higher.

2.3 Genetic Algorithm

Genetics algorithm was implemented in my study as a steady state, spatial and microbial genetic algorithms, this means only one individual genotype in the population will be mutated. The fitness function remains the same (except in this example the capacity changes and differs from the capacity used in hill climbers). The genetic algorithm also features tournament selection which works by randomly selecting two individuals and comparing them to find a winner (W) and loser(L), then the W copy over L and a mutation is added to the loser. I expect these genetic algorithms to have performances better than that of a hill climbing algorithm. The idea of spatial genetic algorithm is to implement local neighbourhood selection which means the G1 can only ever be compared with G2 which must be in the range of k. This ensures the chosen G1 can only ever do comparison and mutation with its neighbours hence creating small sub-populations within the population.

2.4 Steady State Genetic Algorithm with tournament selection Pseudo Code

1. *Initialise a random population P*
2. *For every generation to 100*
3. *Pick 2 individuals at random & evaluate them to find a winner (W) and loser(L)*
4. *Copy W over L and add a mutation to the loser*
5. *Until success or give up, go to 2*

2.5 Spatial Genetic Algorithm Pseudo Code

1. *Initialise random population P*
2. *Assign each individual with a position p*

3. *Pick one individual at random to make it p1*
4. *Pick a second individual p2 in the local neighbourhood of the first according to k*
5. *Compare p1 and p2 finding a winner (W) and loser (L)*
6. *Copy W over L and add a mutation*
7. *Until success or give up, go to 3*

The idea of spatial genetic algorithm is to implement local neighbourhood selection which means the G1 can only ever be compared with G2 which must be in the range of k. This ensures the chosen G1 can only ever do comparison and mutation with its neighbours hence creating small sub-populations within the population.

2.6 Full Microbial Genetic Algorithm Pseudo Code

The inventor of the microbial genetic algorithm Harvey (2001), implemented the recombination operator to the previous algorithm. A value of Pcrossover, which is the probability of crossover will determine the chance that some parts of the gene will be copied. This is different from all the algorithms mentioned before, in that it has utilised combining genotypes with regard to the probability of crossover rather than copying over the genes of W to L.

1. *Initialise random population P*
2. *Associate each individual with a position p*
3. *Pick one individual at random to make it p1*
4. *Pick a second individual G2 in the local neighbourhood of the first according k*
5. *Compare p1 and p2 finding a winner (W) and loser (L)*
6. *Copy each gene of the winner W to the L with crossover (PCrossover)*
7. *Add a mutation according to the loser (L)*
8. *Until success or give up, go to 3*

3 Results

3.1 Single Hill Climber

The single hillclimber was made according to the methods I described in the method section.

I found the graph for a single hill climber to be often stuck on the value of 0. This is the result of the volume of the randomly generated genotype to be greater than the capacity. Occasionally, the randomly generated genotype produces a certain fitness and sometimes can mutate and grow in fitness such as the result shown in figure 5. I found that occasionally it can generate the maximum fitness possible (with hindsight) but often it gets to a point where the line flattens out meaning there are no more mutations that will generate a better fitness. This is known as reaching a local maxima where the individual has reached the peak of the particular region in the fitness landscape.

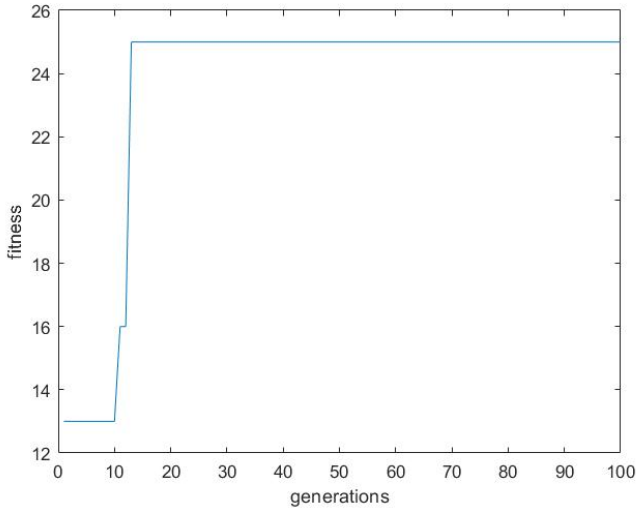

Figure 5- a single random hill climber using the first benefit and volume set. Showing the single hill climber starting with fitness of 13 and taking around 10 iterations before the algorithm finds a higher fitness combination of genotype. Then it is evident that the hill climber got stuck at 25 and cannot increase in fitness anymore, displaying that it must be stuck in a local maxima.

From the results obtained from a single hill climber, it illustrates the chance of the hill climbing algorithm reaching the global maxima is of a relatively low chance. Therefore, I have implemented another algorithm described in the method section above which tries to solve the problem by having a population of hill climbers to simultaneously solve the problem.

In figure 6, I ran a population of 100 individual hill climbers to find the best fitness. As mentioned in the methods, the main reason to implement a lot of hill climbers is to maximise the chance that at least a few hill climbers will reach the global maximum. Figure 6, confirms my method has successfully reached the global maxima. At least 2 hillclimbers have reached the maximum fitness value of 33. Figure 6 shows another problem in which the hill climbers are prone to be stuck on the local maxima.

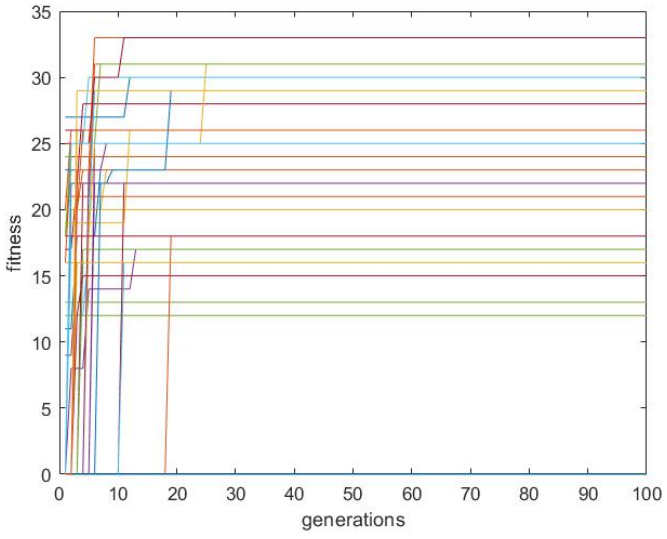

Figure 6 a population of individual hillclimbers
It can be seen that a wide range of performances when generating a population of individual hill climbers.
Also to note, it is very evident that most individual hill climbers tend to get stuck at local maxima.

3.2 Effect of population size

During the study, I discovered that the population must influence the fitness values obtained. I made the populations of hill climbers be able to generate individual lines to display on the graph with a legend showing what the lines represented.

Figure 7 shows a graph that illustrates the different average population size. The graph clears shows an overall trend that greater population size means faster convergence than smaller population size. With the population of 10 being the line with the highest average fitness value. This supports my hypothesis in that, higher population sizes will give better overall hill climbing results.

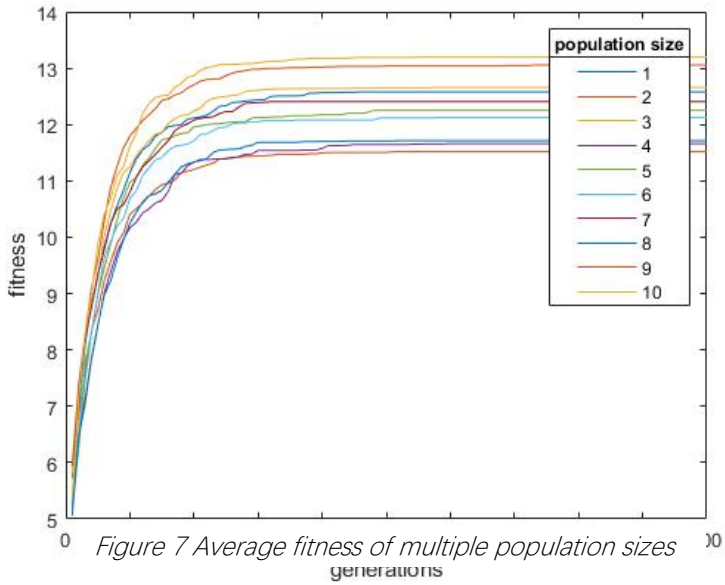

Figure 7 Average fitness of multiple population sizes

3.3 Steady State with Tournament Selection Genetic Algorithm

Figure 8 shows the average (mean) fitness value of an individual hill climber and the average (mean) fitness value of a steady state genetic algorithm with tournament selection growing over 1000 generations. It is interesting to note that the individual hill climber has a low starting fitness value of around 3.3 whereas, the fitness of the steady state is around 10. There is a major difference in their starting fitness value. However, the individual hill climber can climb more rapidly compared with the gradient of the steady state line. It is also worth noting that the individual hill climber seems to be stuck on the local maxima and the steady state is

continuing to grow. The best fitness reached after 1000 generations is clearly that of the steady state with tournament selection.

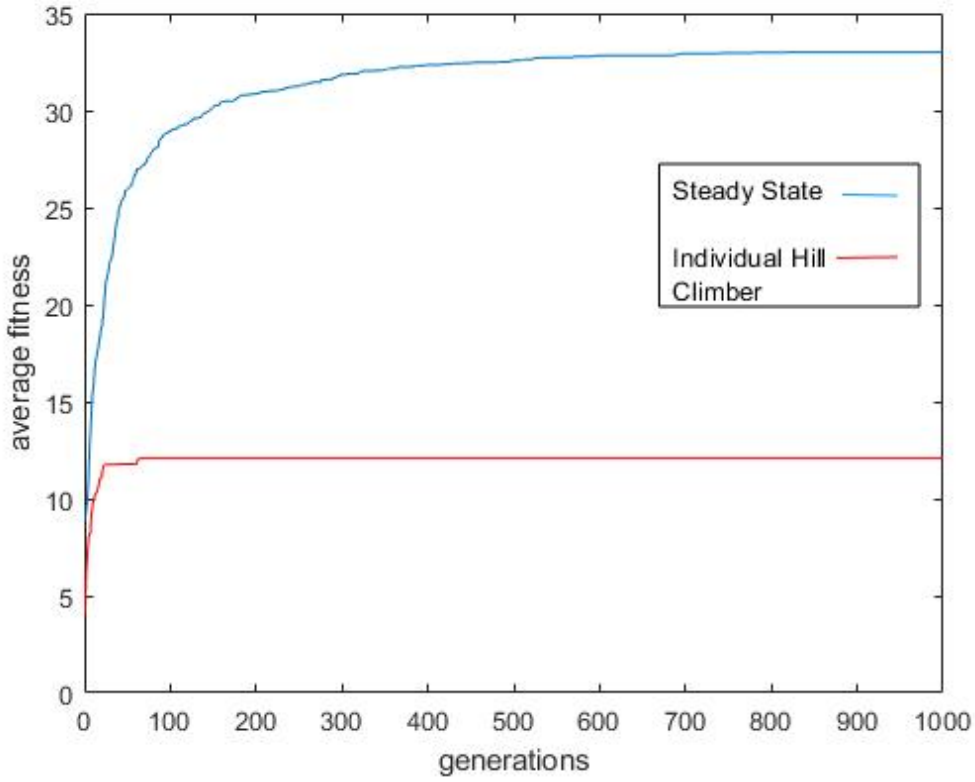

Figure 8 Average fitness of Steady State GA Vs Individual Hill Climber over 1000 generations, Steady State were changed so that it used the same sets of benefits and volumes.

It is evident that the average fitness of the individual hill climber does not exceed 13 fitness values whereas, given enough generations, the steady-state genetic algorithm could reach the global maxima. This suggests given enough generations eventually the most optimal solution will be found with steady state and some hill climbers will be stuck on a local maxima which hinders the average fitness value. **Important to note**, individual hill climbers could reach the best optimal fitness value but not the average.

3.4 Effect of PCrossover values

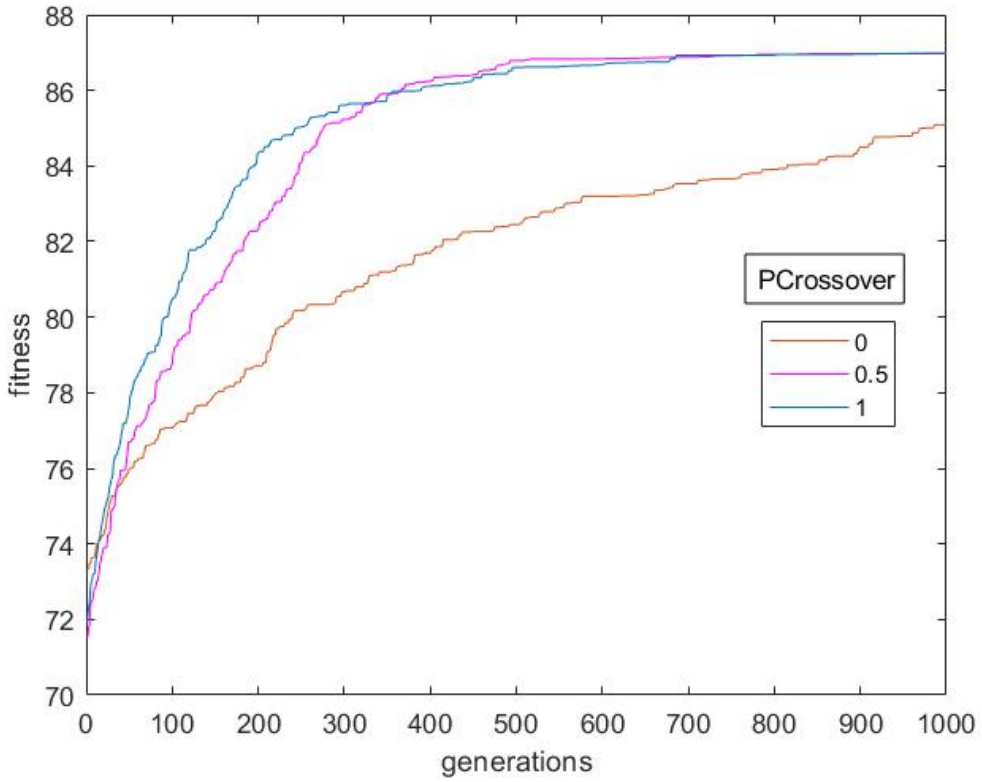

Figure 9 Average fitness over 1000 generations of Microbial GA with different PCrossover values The graph displays the average fitness of the population at each generation. Using the average provides a fair way to compare the fitness between the different PCrossover values.

Figure 9 shows that when the probability of crossover is at 0, where there is no copy over of the genes from winner to the loser, the loser simply mutates and this significantly hinders the population of genotypes from improving their fitness values. When the probability is set at 0.5, around half of the times, the loser obtains the genes from the winner, this creates more variation and will increase the chance that the genotype will have better fitness values. Finally, the best result is from the PCrossover value of 1 where the loser copies every gene from the winner and mutates, it allows the overall fitness to be higher than other PCrossover values because, at each iteration, the loser would have a better chance of having better fitness than the winner after mutation. So when the probability is set to 1, the overall fitness would converge faster to the global maxima than any other PCrossover values. This is the result of eliminating the genotypes with worse fitness values from the population.

3.5 Spatial Genetic Algorithm Vs Full Microbial Genetic Algorithm

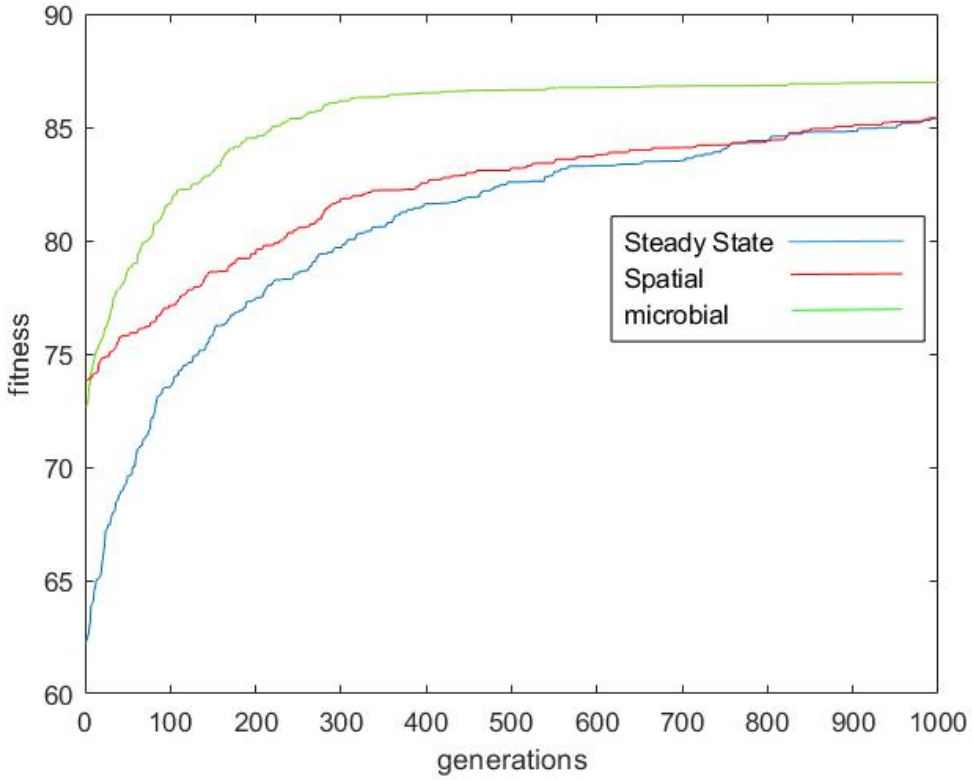

Figure 10 Average over 1000 generations of Steady State vs Spatial vs Microbial GA comparison between the performances between the three type of genetic algorithm, microbial is the best performing algorithm followed by spatial and worst performing algorithm is steady state.

Figure 10 shows that the best performing genetic algorithm is the microbial genetic algorithm, and this is to be expected because the most basic genetic algorithm is the steady state, it has only tournament selection to compare two randomly chosen genotypes, the spatial genetic algorithm is a slightly improved version in that it only compares with the nearest neighbours and the more sophisticated microbial genetic algorithm adds on top of the previous improved methods and adds a PCrossover and from the previous 3.4 investigations, PCrossover value of 1 is the best performing so when using full microbial genetic algorithm with PCrossover of 1, the result is the best. Firstly, the spatial genetic algorithm performs better than steady state algorithm proves that comparing the nearest neighbours allows faster convergence to the global maxima, I think this is because choosing within the neighbourhood helps to stop the population becoming panmictic and instead allows the dispersion of demes, this allows more diversity across the population. The other factor making microbial genetic algorithm has good performance is the PCrossover, it determines how much of the winner's genotype is copied to the loser, this instantly allows the loser to gain much higher performance than with other algorithms so it is much more effective at increasing the overall fitness value across the population.

4 Discussion

Initially, all the different algorithms did only one run which means the results differed greatly due to the random nature of the

genotype generated. To combat this problem, I implemented another for loop for every algorithm and worked out the total fitness and divided by the total number of runs to get the average (mean) of the algorithm. This provided a far worthier graph to be discussed and compared for this scientific study.
I found that my hypothesis was supported by my study. It is very evident that the individual hill climber performed considerably worse than the population of hill climbers and from the theoretical point of view, this is to be expected. When an individual hill climber is attempting to reach the maxima, it may be hindered by local maxima, whereas a population of hill climbers will increase the likelihood of some genotype reaching very high fitness hence increasing the average value.

Local maxima were the main cause for the relatively poor performances by the hill climbers because when the hill climbers are placed in the fitness landscape randomly, they were unable to mutate enough to realise that there are other higher fitness hills in the landscape. Being stuck at the local maxima, the hill climbing algorithm provides no method for the improvement of the individual hill climber hence the fitness value will not increase after a certain number of generations. However, this is not a problem for a more advanced genetic algorithm such as microbial genetic algorithms. Genetic algorithms are able to navigate around the landscape as they have the ability to find genotypes with better fitness and the genotypes with worse fitness values will copy their genotypes and mutate so there is no problem with being stuck on sub-optimal maxima.

The effect of the number of generations on hill climbers and genetic algorithm is very profound. The hill climbing algorithm seems to be stuck on average fitness levels and cannot improve its fitness even if the number of generations is increased by a factor of 10 (1000 generations). The genetic algorithms have shown that even after 100 generations, it continues to improve its fitness value but of a lesser rate, however, this is still good to see that only after around 500 generations, the average global maxima have been reached. I could also expand this section of the investigation by finding out the average maximum fitness (obtain the best fitness value of each generation but run the algorithm in parallel multiple times and find the average) at each generation rather than the average fitness, this would allow the fitness values for hill climbers to reach global maxima.

Also, I could improve my study by having the most optimal mutation rate to compete with the steady state with tournament selection.

This study illustrated the different algorithms for solving an optimisation problem and how various parameters such as PCrossover, size of a population affects the performance of each algorithm. Also, interestingly, the biologically inspired microbial genetic algorithm had the best performance showing that real-life examples could help solve problems in computer science and in the grand scale of things, it is interesting to see how nature is an optimisation problem and links with optimisation problem on the computer

5 References

Rechenberg, I., 1964, September. Kybernetische lösungsansteuerung einer experimentellen forschungsaufgabe. In Annual Conference of the WGLR at Berlin in September (Vol. 35, p. 33).

Fraser, A. and Burnell, D., 1970. Computer models in genetics. Computer models in genetics.

Yang, X. (2006). An introduction to computational engineering with Matlab. Cambridge: Cambridge International Science Pub.

Yu, X. and Gen, M. (2010). Introduction to evolutionary algorithms. London: Springer.

Uk.mathworks.com. (2019). What Is the Genetic Algorithm?- MATLAB & Simulink- the MathWorks United Kingdom. [online] Available at: https://uk.mathworks.com/help/gads/what-is-the-genetic-algorithm.html [Accessed 20 Mar. 2019].

Harvey, I (2001) The Microbial Genetic Algorithm. Centre for Computational Neuroscience and Robotics, University of Sussex, Brighton

Mitchell, M., Forrest, S., & Holland, J. H. (1992, December). The royal road for genetic algorithms: Fitness landscapes and GA performance. In Proceedings of the first European conference on artificial life (pp. 245-254).

Holland, J. H. (1962). Outline for a logical theory of adaptive systems. Journal of the ACM (JACM), 9(3), 297-314.