Artificial intelligence cw

# Abstract

# Introduction

## Task1: A

To be able to solve the continuous optimization problem using sum of squares, the Genetic Algorithm(GA) would need to be created. A typical GA is made up of 6 main components these being: Initialisation of the population, evaluation, selection, crossover, mutation, and survival. These will allow the GA to function as once the population is established it will repeat all the steps from evaluation to survival over and over again which would be defined as a generation. This will continue to occur until the solution for the problem is found within any of the genomes of the population or if the amount generation which have occurred has exceeded the generation limit. The limit is used so that the GA does not try and solve the problem for an infinite amount of time which may occur if for some reason a solution is unobtainable.

To generate a population, the GA would need to take in the multiple parameters which will define the population. GENE\_LENGTH dictate the number of genes per genome in the population. Both the UPPPER and LOWER\_BOUNDs dictate the range of values which will be spawned into population. POPULATION\_SIZE defines the amount of genome which are spawned. All of these parameters are used to create the initial population. Each of the parameter can be changed depending on what problem is being solved or how it is tested. From this, it will create a population with random values between the two bounds with each genome having an amount of these values up to the GENE\_LENGTH

Once the initial population has been established, the fitness of the population needs to be calculated. This is done so that the GA can identify which of the genomes is closest to finding the solution or if a genome has already found the solution. The fitness algorithm used is the Sum of Squares as that is the problem that is being solved. It will process each of the genomes through the sum of square formula which will produce a value which it that genomes fitness. Due to this being a minimisation problem, the closer the fitness is to 0 the more valued that genome is in the population.

Once these values have been found in the evaluation stage, the selection stage can be processed. This is done to selected the parents for the next generation in population which is done based off the fitness of the current generation. The goals is to have the populations fitness to improve though adequate selection. This can be done in multiple forms as there are lots of different selection method like roulette wheels selection which assigns a proportional probability to each of the genomes of the population and does it’s selection randomly. For this GA, it will use rank-based selection. This is where the selection process is done by ordering all the genomes in the population from best fitness to worst fitness. This means that the best fit genomes will be selected to crossover with each other which should lead to their best part being crossed over leading to the next generation have a lower fitness. This method is better and improving already well performing genomes while providing very little to genomes which have a bad fitness score as they will only be crossing over with other genomes with bad fitness.

With the parents selected, the next generation needs to be created. This is done through a crossover operation where parts of each parent are spliced together to create the offspring which become the next generation. The way in which the offspring are formed can be done in multiple ways and for this GA one-point crossover is used. In one-point crossover, a point is randomly chosen in the genomes and then both parents are spilt at the chosen point. This creates two split genomes from each parent. Then each respective part from each parent is combined with the other part from the other parent. This would then create two new genomes from the parents which would become the part of the new generation. To control how often the occurs within the GA, SELECTION\_RATE is used. This dictated the likelihood of this operation occurring. If it does not occur the parent are unchanged and the children sent to the next generation become clones of each respective parent. Doing this for the entire population will create the next generations population. Crossover occurs as the GA want to combine the genes of the best genomes in the population to hopefully create new genomes with even better genes than it’s parents. Along with this is can help with passing gene which are considered better fit for the problem to other genomes in the population which should improve the population overall fitness towards the problem.

Once the all the children have been created, they are then put through mutation. This process has a chance to change the composition of each genome in the population. The rate of this is controlled and dictated by the MUTATION\_RATE. The way in which the mutation occurs within this GA is with a random bit assignment. This will take a random gene within the genome being mutated and will change it to a random value. The reason this is done create mutation which may become beneficial to the population and potentially overwriting genes which have a poor fitness. However due the random nature of mutation It can also overwrite genes with good fitness for ones with worse fitness. Another benefit to mutation is that is the desired genes to solve problem do not exist population, mutation can lead to it being introduced into the population allowing for the solution to be feasible.

With all of these functions of the GA being implemented it allows for the Sum of Square function to be solved along with others.

## Task1: B

The GA was then expanded so that if could solve more problem which are also continuous optimisation problem. The other problem which were chosen to be solved are: Dixton and price, Levy, Zarkhov ,Perm and Ackley. For each of these the parameter were change to accommodate for these problems. Along with that their would fitness algorithm was introduced into the GA.

## Task1: C

To test the performance of the GA against the problems chosen, the parameters of the GA were changed to see how it affected the performance. The performance was measured by the mean amount of generations it takes for the GA to find the result to each problem. Along with this is included the lower and upbound for generation taken at each test case. To find the Mean generations the algorithm would be run for set amount of times with the mean of the sum of the results producing it’s performance. It would start from the base setting of population size = 50, gene length = 5, mutation rates = 0.2 and crossover rate = 0.8. these are also the parameters which will be adjusted for testing

### Sum Of Squares

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Population Size** | **Length of Genes** | **Mutation Rate** | **Crossover Rate** | **Least Generation** | **Most Generation** | **Mean generation** |
| 50 | 5 | 0.2 | 0.8 | 49 | 7941 | 2113 |
| 50 | 5 | 0.05 | 0.8 | 203 | 2155 | 737 |
| 50 | 5 | 0.5 | 0.8 | 4870 | 11693 | 8213 |
| 50 | 5 | 0.2 | 1 | 96 | 2994 | 1177 |
| 50 | 5 | 0.2 | 0.5 | 3109 | 22869 | 11502 |
| 100 | 5 | 0.2 | 0.8 | 225 | 2574 | 1003 |
| 10 | 5 | 0.2 | 0.8 | 99 | 91102 | 42657 |
| 50 | 10 | 0.2 | 0.8 | N/A | N/A | N/A |
| 50 | 3 | 0.2 | 0.8 | 4 | 232 | 53 |
| 100 | 5 | 0.05 | 1 | 39 | 116 | 67 |
| 125 | 10 | 0.025 | 1 | 134 | 548 | 336 |

From this, it shows the generals factor which increase performance are the increase in population size, the decrease of mutation rate in relation to the amount gene and population, the increase in the crossover rate. Knowing this, allows for the GA setting to be modified appropriately to tackle versions of this problem with greater complexity due to gene length in a genome.

### Dixon and Price

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Population Size** | **Length of Genes** | **Mutation Rate** | **Crossover Rate** | **Least Generation** | **Most Generation** | **Mean generation** |
| 50 | 5 | 0.2 | 0.8 | 482 | 52207 | 19891 |
| 50 | 5 | 0.05 | 0.8 | 78 | 9658 | 2105 |
| 50 | 5 | 0.2 | 1 | 513 | 16578 | 4902 |
| 100 | 5 | 0.2 | 0.8 | 435 | 11712 | 5902 |
| 50 | 3 | 0.2 | 0.8 | 4 | 227 | 84 |
| 100 | 5 | 0.05 | 1 | 30 | 417 | 148 |
| 125 | 10 | 0.025 | 1 | 1144 | 36609 | 10542 |

As with sum of square, the performance of the algorithm would increase change to the crossover rate, mutation rate and population size. For Dixon and Price it seems that the mutation rate had the greatest effect on the overall performance of the algorithm. This could be because a lower mutation rate reduces the chance of randomness in the GA were genes are changed. It also seems to scale worse as when the number of genes is increased it took much longer under the same parameters as the sum of squares.

### Levy

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Population Size** | **Length of Genes** | **Mutation Rate** | **Crossover Rate** | **Least Generation** | **Most Generation** | **Mean generation** |
| 50 | 5 | 0.2 | 0.8 | 36 | 2839 | 1245 |
| 50 | 5 | 0.05 | 0.8 | 85 | 953 | 338 |
| 50 | 5 | 0.2 | 1 | 28 | 3611 | 1258 |
| 100 | 5 | 0.2 | 0.8 | 22 | 2889 | 1386 |
| 50 | 3 | 0.2 | 0.8 | 0 | 111 | 56 |
| 100 | 5 | 0.05 | 1 | 19 | 110 | 59.2 |
| 125 | 10 | 0.025 | 1 | 148 | 627 | 304 |

### Zakharov

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Population Size** | **Length of Genes** | **Mutation Rate** | **Crossover Rate** | **Least Generation** | **Most Generation** | **Mean generation** |
| 50 | 5 | 0.2 | 0.8 | 12020 | >100,000 | N/A |
| 50 | 4 | 0.2 | 0.8 | 169 | 8639 | 2484 |
| 50 | 4 | 0.05 | 0.8 | 578 | 6445 | 2119 |
| 50 | 4 | 0.2 | 1 | 674 | 6590 | 2870 |
| 100 | 4 | 0.2 | 0.8 | 11 | 6127 | 1747 |
| 50 | 3 | 0.2 | 0.8 | 3 | 542 | 219 |
| 100 | 5 | 0.05 | 1 | 830 | 43618 | 7166 |
| 125 | 10 | 0.025 | 1 | N/A | N/A | 100,000> |

From this, it shows how for Zakharov the factor which seems to affect the performance the most is the population size as it had the most dramatic effect on the average though general mean. Also the amount of genes in each genome had a dramatic effect on the performance as from going from just 5 to 4 genes in a genome the performance suffered immensely and even with parameter changed it is unable to solve it with a gene length of 10 unlike some of the other problem.

### Perm

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Population Size** | **Length of Genes** | **Mutation Rate** | **Crossover Rate** | **Least Generation** | **Most Generation** | **Mean generation** |
| 50 | 4 | 0.2 | 0.8 | 233 | 2995 | 1758 |
| 50 | 4 | 0.05 | 0.8 | 120 | 4022 | 1546 |
| 50 | 4 | 0.2 | 1 | 206 | 4098 | 1717 |
| 100 | 4 | 0.2 | 0.8 | 107 | 3470 | 1201 |
| 50 | 3 | 0.2 | 0.8 | 12 | 311 | 157 |
| 100 | 5 | 0.05 | 1 | 68 | 2755 | 1230 |
| 125 | 10 | 0.025 | 1 | N/A | N/A | N/A |

From Perm , it shows immediately that the scaling into large problems to solve is an issue as it would struggle to get result within a reasonable amount of generation. It also seems that it is more important to change the parameters appropriately to the size of the problem when solving the problem. It also shows that when the parameters are changed it has overall a small difference in its ability to solve the problem but does generally improve it. Compared to others it seems more reliant on changed the parameters to meet the length of the gene.

### Ackely

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Population Size** | **Length of Genes** | **Mutation Rate** | **Crossover Rate** | **Least Generation** | **Most Generation** | **Mean generation** |
| 50 | 4 | 0.2 | 0.8 | 1045 | 9646 | 5655 |
| 50 | 4 | 0.05 | 0.8 | 749 | 6317 | 2285 |
| 50 | 4 | 0.2 | 1 | 554 | >10,000 | 3146 |
| 100 | 4 | 0.2 | 0.8 | 260 | 6426 | 2624 |
| 50 | 3 | 0.2 | 0.8 | 41 | 2354 | 878 |
| 100 | 5 | 0.05 | 1 | 102 | 3962 | 1356 |
| 125 | 10 | 0.025 | 1 | 292 | 3433 | 743 |
| 175 | 10 | 0.025 | 1 | 161 | 519 | 285 |

# Task1: D

Along with test how the parameters affect the performance of the GA, other method can be used to affect the performance of the GA. To do this another selection method is used. The alternative method to be used is the roulette selection method. This method assigns a set value to each of the genomes in the population which is proportional to the level of fitness which they have. This is the probability of the genome being chosen as the parent. This should change how population evolves is it won’t be just the best fit genome crossing over with the other best genome. This should increase the overall fitness of the population. Below are the results from sum of squares going through roulette selection:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Population Size** | **Length of Genes** | **Mutation Rate** | **Crossover Rate** | **Least Generation** | **Most Generation** | **Mean generation** |
| 50 | 5 | 0.2 | 0.8 | 5 | 24 | 14 |
| 50 | 5 | 0.05 | 0.8 | 8 | 38 | 17 |
| 50 | 5 | 0.2 | 1 | 7 | 27 | 14 |
| 100 | 5 | 0.2 | 0.8 | 9 | 16 | 12 |
| 50 | 3 | 0.2 | 0.8 | 3 | 15 | 8 |
| 100 | 5 | 0.05 | 1 | 9 | 19 | 13 |
| 125 | 10 | 0.025 | 1 | 15 | 28 | 21 |
| 125 | 50 | 0.025 | 1 | 101 | 183 | 124 |

The result show that with random escalation the performance of the GA is dramatically improve. Where in rank selection it would 1,000s of generation to solve the problem, with roulette it solved much faster. Even though change the parameter has a less dramatic effect on the results as with Rank, they still do improve the performance even to a more subtle degree. Along this roulette seems a lot more robust when dealing with different size of gene length as with a large difference in size from 10 to 50 genes the performance was only mild hurt, where in comparison rank would struggle with a slight increase in gene length size.

# Task2: A

Combinatorial optimisation problems are when the variables used to fill up a genome are limited to a range of selected values. To create a GA which accommodates this, it uses most of the basic structure from the continuous distribution problems. For a problem such as the sum of the 1s in binary string, you can use it as either a maximisation problem where the answer is all 1s while for minimisation you would get the solution of 0s. For this problem the GA has been made to find the minimisation of the problem. The method used to calculate the fitness was by taking the sum of all the values in the binary string which would produce a fitness value. This works well as the only variable that can be added to the genes are 1 and 0 and if the it’s tying to solve for minimisation the goal would be 0 so a sum of the genes would produce an accurate fitness. Along with modifying the fitness algorithm, generating population and mutations are changed so that the values which are added can only be from the variable set VARSET.

# Task2: B

The other Combinatorial optimisation which I implemented are N – Queue, map colouring and Knapsack.

For knapsack the goals is to be able to carry the most amount of object with the most value without become unencumbered. The items are generated at the beginning of the algorithm and the amount generated is defined by the length of the genes used. This is because it uses a binary string of 0s and 1s to represent the items with genome with 1 represent the item is in the genomes sack while 0 represents the lack of the item. With the items that can be added there is an associated weight with them. This is also randomly generated and uses weights between a range which can be set. The weight is used to dictate how many items and which items can be carried by each genome. This is because the genome must abide by a weight threshold. This means that each genome cannot carry items whose total weight would exceed the threshold. The plays a large part withing the fitness algorithm. In this it takes the sum from each value of each item in a genome to use as the fitness for the whole genome. It also gets the sum of the weight of all the item within the genome and if the sum of the weight would exceed the threshold for the genomes it will give the poorest fitness. This will ensure that it does not crossover with genome with good fitness. Due to the nature of the problem, it is a maximisation problem as the goal is to get the greatest value of items from the list available and space given. This will mean in selection the genome with the highest value are those with the highest fitness.

This problem does not have a definitive solution to the problem as even a genome which is full of item up to the threshold may not be the solution. This is due to the values which are associated to each of the item meaning that it is not as simple as being able to fill the genome to the top with item. To find a solution, a limit has to be introduced so that it does not go on infinitely. This GA tracks which solution has been found with the best fitness and if a solution continues to be the best solution for more generation than the threshold the GA will end and present the current best solution as the answer. The parameter needs to be large enough to be able to give enough generation to see is another solution is created.

For the N – Queen problem, it is trying to solve the problem of there being n queens on a n\*n chess board where none of the queens can possibly take each other. This was implemented into the GA so that it will find the solution to this problem since it is a minimisation problem. This is due to the goal being getting 0 hits(being instances where a queen can be taken by another).

To create a population to solve this problem, each genome must have a variable within N, N being the amount of queen to be used/size of the boar, so in a problem of N = 5, a genome can only use the values of {0,1,2,3,4}. The amount of genes in the genome is also of size N. this allows for a genome to represent the entire board of N\*N and the positions of the queens.

To calculate the fitness for this problem, the amount of hits was the factor in fitness value. This was done be calculating sum of the amount of hits from all the queen in a genome. This then allows for the selection of the best parent to be done on which genome obtained the lowest value and therefore the highest fitness. The normal GA process occurs until and answer reaches 0 fitness being the solution to the problem. For each board there are multiple solution to this problem so on each run of the GA different solution can be found.

In the map colouring problem, the goal is to be able to fill in a map made from defined quadrants with N colours with no colours repeating for any of the quadrants neighbours. To implement this into the GA a list of neighbour is created per quadrant. This contains each neighbour on the map an allow for the correct neighbour to be checked when calculating fitness. The goal of this problem is to get no conflicts so it is a minimisation problem where the goal is 0 fitness. When creating the population for this, the length of the genome is set to the number of quadrants on the map, which each gene in the genome being from the list of N colour variables. To calculate the fitness of each genome, each gene is checked to see if it conflicts with a neighbour and if so makes the fitness for that genome worse.

# Task3: A

The real world problem the this GA has been designed to tackle is the traveling salesman problem. This problem is defined that with N cites, the salesman wants to find the fastest route to travel to all of the cities once and return to where they started. This problem would be a minimisation problem as the goal is to produce and answer which is the smallest distance need to travel to traverse all the cites.

To generate the population for this problem, each gene in a genome is given a value which with within the range of the amount of cites. Along with this each city value can only be assigned once to the genome expect the origin city which it both the first and last gene within every genome. With this each genome should have their own unique route around all the desired cities with each gene representing the next city to travel too.

To calculate the fitness of each route, weights need to be introduced. These are assigned to each city and represent the weighted value of traveling from that current city to another within the problem. These are all randomly assigned values with the values going to the themselves always being set to 0. It then uses these weights to assigned weighted value to route travelled and the sum of becomes the fitness for that genome. Along with this, the fitness function also checks the contents of the genome to see if the values would create a complete route to visiting all the cites. If the genome does not contain the required values, it give the genome a bad fitness score ensuring that it will be at the bottom of the fitness ranking.

Like the knapsack problem, because there is no one defined end goal for the algorithm, the current best fit genome is store and is not replaced by a certain threshold will stop the algorithm and present the current best fit as the solution.

# Cd;cd

A GA optimising can be used within a Neural Network(NN) for the purpose of optimising it. Generally a GA works best with a NN which is using unsupervised learning. This is because unsupervised algorithms is teach the NN to learn untagged data through learning the data given and GA is much faster way to train the weights to learn the problem faster. This is because the weight in a NN can be broken down into a chromosomes for a GA which allows the weight to be able to be trained by the GA. This benefits a NN as it mean that’s the NN should have better trained weight leading to result becoming more accurate much faster. This made better as due a GA not being very computational expensive as there is less to calculate within a GA allow to repeat the learning constantly without being to resource expensive. Another benefit is that by having the weight learning done through a GA it makes it much easier to be able to understand the progress the GA makes to come to the solution as it will show each step of how the algorithm evolves.

One of the disadvantages of using a GA is that due to factors like the mutation and crossover rate, it can take a long period of time. This is because on generation with poor probability can lead to work being undone within the GA which leads to more time needed to solve the problem. The variance introduced by can be unappealing if consistency is desired for the NN.