**Genetic Algorithm Optimization**

**Ethan David Donovan (19016318)**

**1 INTRODUCTION**

Here in this report, we will be looking into a stochastic search optimisation problem-based nature. We will be looking into the biology of the algorithm and comparing it to a genetic algorithm. We will also be implementing different variations of parent selection, mutation, and crossover with different minimisation functions to test & compare the performances to see which one produces the best results and why they do so. We will do this by implementing a genetic algorithm and adjusting the hyper parameters to see how they affect the final output of the algorithm.

**2 BACKGROUND RESEARCH**

In this section of the report, we will be discussing a meta-heuristic algorithm that is inspired by nature. The algorithm we will be looking at is the Butterfly Optimisation Algorithm (BOA). This is algorithm mimics the behaviour of how butterflies forage for food. Using their antennae, butterflies have a highly accurate sense of smell, sight and taste that is used for mating and foraging. A butterfly emits a fragrance which has an intensity that equates to the butterfly's fitness. For example, if a butterfly moves from one location to another, its fitness with change appropriately (Arora, Singh, 2019). A butterfly uses the fragrance emitted by other butterflies to move around in a given space, they will either move randomly when no fragrance is detected, also called local search, or will move towards a butterfly with the greatest fragrance, also known as global search, this fragrance will equate to a butterfly’s fitness.

There are three important concepts to understand about a butterfly modality to know how fragrance works, these are the sensory modality, stimulus intensity and the power exponent. A modality is a sensory input which can be sound, smell and sight. So, a sensory modality would be what the butterfly perceives after these stimuli like sound or smell. For example, the sound modality would be stimulated when the there is a pressure difference in the air, the ear drum would be stimulated. The stimulus intensity refers to the how powerful another butterfly’s fragrance is. The higher the intensity of the fragrance the closer the butterfly is to it (Guo, Liu, Chen, 2021). Finally, the power exponent is how intense the fragrance is. The higher the fragrance that is generated from a butterfly the higher the chance a butterfly in the surrounding area will be attracted to it (Arora, Singh, 2019). To calculate the fragrance, we use the aforementioned features of a butterfly.

*F = cIa*

Equation 1: Fragrance Calculation

Where *F* represents the strength of the butterfly’s fragrance. *C* represents the sensory modality which has the parameters of *C є [0, ∞].* *I* represents the stimulus intensity and *a* isthe power exponent which can be between *a є [0, 1]* (Guo, Liu, Chen, 2021). Next there are 2 important parts to the algorithm, they are the global search and local search.

The Global search involves a butterfly moving towards another butterfly/solution with the higher fitness which is expressed as:

*Xit+1 = Xit + (r2 × g \* - Xit) × Fi*

Equation 2: Global Search Calculation

*Xi* represents the *ith* butterfly in the iteration. *Xit+1* represents the next butterfly over in the iteration. *t* represents the number of iterations. *g\** represents the best solution in that generation among other solutions. *Fi* represents thefragrance of a butterfly and *r* represents a random number of [0,1] (Arora, Singh, 2019).

Local search is where a butterfly will randomly search a problem space to find a more accurate solution.

*Xit+1 = Xit + (r2 × Xjt - Xkt) x Fi*

Equation 3: Local Search Calculation

*Xjt* & *Xkt* represents different individual butterflies in the same population. All other expressions still apply from global search to local search.

Now that we have discussed the different equations that are used to make BOA work. Below in figure 1 is the pseudocode for BOA. We can now start to look at how the Butterfly Optimization Algorithm compared to a standard genetic algorithm seeing how they each compare against each other with different fitness function.

When comparing the BOA with a GA there are a few things that stand out. A genetic algorithm searches a problem space by using mutation and crossover. The global search mechanism in a GA would be the crossover function. The reason for this is because crossover can make large jumps in individual’s values to bring it closer to a solution. The local search mechanism would mutation. This is because it makes smaller changes to an individual which allows it to search in its local space to find the best solution. BOA also has 2 search mechanisms.

BOA uses global search to find other butterflies with a high fragrance, this is similar to crossover because it will make bigger changes in the values to get close to a solution. BOA’s local search is random, this is somewhat like mutation in how it will randomly adjust values however mutation is base on a chance that a value will change and is not completely random in where an individual will search.

**3 EXPERIMENTATION**

A Genetic algorithm starts with population initialisation, this is where a population is initialised with a set of individuals with random values. Then that populations fitness is then evaluated to be compared with later in the algorithm. Next, we then start a loop which will iterate over the following. The first part is selection, this is where parents are selected from a population that will be used for breeding later. Next, we then crossover different parents to create offspring, then those offspring are then randomly mutated to generate new values, then we evaluate the offspring’s fitness and then select the best individuals to be carries over to the next generation.

Here in this section, we will start to look and compare different selection, crossover, mutation methods testing against 2 different fitness functions which are the Rosenbrock and Ackley function seeing how they stack against each other. The format of each test will be layout as so. First discuss what functions we will be testing, then show the different parameters that will affect the results, show the results in a line graph which are ran 5 times and then we take the average of the 5 and then finally discuss how the parameters change the results. There are 5 main parameters that we will be looking at in each test. These are:

* Epochs
* Population size
* Gene size
* Mutation rate
* Mutstep
* Extra: Gap size

We will be keeping the Epochs, Population size and Gene size parameters the same to make sure there is some consistency for each test, so they are comparable. We will be performing a parameter sweep for the Mutation rate and Mutstep so we can refine and optimises the algorithm to try and reach the global optimum for each fitness function.

3.1 Test 1

For the first test we will be looking at Tournament Selection, Single-point crossover and mutation. We are using the Rosenbrock fitness function to see how changing the mutation step and Mutation rate will affect the algorithm. To find the best parameters for this set of functions and the Rosenbrock function we performed multiple parameter sweeps. The parameters for this test will be

* Epochs: 500
* Population size: 50
* Gene size: 20

Iteration 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Mutstep\  Mutation Rate | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
| 5 | 856.56 | 7353.59 | 3812.60 | 52587.42 | 57427.48 |
| 10 | 2568.91 | 44234.31 | 215253.89 | 378882.79 | 393808.04 |
| 15 | 45384.89 | 277748.31 | 1148201.56 | 2182945.30 | 3728993.60 |
| 20 | 43470.55 | 647843.41 | 2804211.14 | 3314924.49 | 10175852.11 |
| 25 | 55710.54 | 1695252.38 | 7995050.10 | 17300841.31 | 37662502.03 |

We started to test a mutstep of 5 with different mutation rates going down. The reason for this is because having a too high of a mutstep means that when the algorithm starts to converge to an optimal solution, it can overshoot the global optimum because it makes bigger changes to the genes. In the table below we did more testing to find better parameters for the algorithm.

Iteration 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Mutstep\Mutation Rate | 0.05 | 0.06 | 0.07 | 0.08 | 0.09 |
| 1 | 252.71 | 405.48 | 314.34 | 326.76 | 260.10 |
| 2 | 511.46 | 149.35 | 232.75 | 358.92 | 412.66 |
| 3 | 168.177 | 535.67 | 456.46 | 336.99 | 461.97 |
| 4 | 479.66 | 460.73 | 221.82 | 329.48 | 559.29 |
| 5 | 380.69 | 461.82 | 320.57 | 998.90 | 653.31 |

As you can see from figure 1 below, we get an average best fitness over 10 system iteration of 149.35 and it starts to converge around 50 generation. The reason why the graph starts to plateau after that is because it is making small changes to the genes in each individual to get closer to a solution.

Chart

Description automatically generatedChart, histogram

Description automatically generated

Figure 1: Mutstep: 2, Mutation Rate 0.06

3.2 Test 2

For the second test, we be looking at Tournament Selection, Single-point crossover and mutation. We are using the Ackley fitness function to see how changing the Mutation Step and Mutation rate will affect the algorithm. Below is a table showing the different parameters that were used to test the fitness function. Here our default parameters are:

* Epochs: 500
* Population Size: 50
* Gene size: 20

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Mutstep\Mutation rate | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
| 5 | -20.11 | -18.16 | -17.13 | -16.33 | -15.58 |
| 10 | -18.36 | -15.46 | -14.11 | -12.53 | -11.75 |
| 15 | -17.37 | -13.54 | -11.15 | -10.38 | -8.15 |
| 20 | -16.25 | -11.74 | -9.14 | -7.69 | -6.82 |
| 25 | -15.23 | -10.45 | -8.01 | -6.06 | -5.04 |

As you can see from the table above, the algorithm does not quite reach the global optimum but gets close. We will perform another parameter sweep to try get a better solution.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Mutstep\ Mutation Rate | 0.05 | 0.06 | 0.07 | 0.08 | 0.09 |
| 1 | -22.701 | -22.695 | -22.694 | -22.693 | -22.686 |
| 2 | -22.688 | -22.687 | -22.678 | -22.674 | -22.655 |
| 3 | -22.661 | -22.666 | -22.651 | -22.639 | -22.599 |
| 4 | -22.659 | -22.637 | -22.633 | -22.621 | -22.569 |
| 5 | -22.642 | -22.647 | -22.612 | -22.604 | -22.445 |

After performing another parameter sweep, we get much closer to a solution. We improved the performance of the algorithm by decrease the mutstep and mutation rate. When comparing figure 2 and figure 3 we can see that they both converge to a solution at different generations. Figure 2 converges around 100 generations but figure 3 converges a just before 200 generations. This happens because with a higher mutstep, you can make bigger changes in the genes to get closer to a solution in less time. To prove this is the case, we will keep the mutation rate at 0.05 and change the mutstep to 5. As shown in figure 4, we can see that the algorithm converges much earlier, around 100 generations much like figure 2 with the same mutstep. This shows that the mutstep has an impact on how early the algorithm will converge to a solution

Figure 2: Mutation Rate: 0.1, Mutstep: 5 Best Individual: -20.117755374919746

Chart

Description automatically generated

Chart, histogram

Description automatically generatedFigure 3: Mutation rate: 0.05, Mustep: 1 Best Individual: -22.701359620362815

Figure 4: Mutation rate: 0.05, Mutstep: 5 Best Individual: -22.642

Chart, histogram

Description automatically generated3.3 Test 3

For this test we will be looking at Tournament Selection, Single-point crossover and gaussian mutation using the Rosenbrock function. Here our default parameters are:

* Epochs: 500
* Population Size: 50
* Gene size: 20

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Deviation\Mutation Rate | 0.2 | 0.4 | 0.6 | 0.8 | 1 |
| 0.2 | 18.876 | 18.885 | 18.906 | 18.898 | 18.957 |
| 0.4 | 18.788 | 18.818 | 18.828 | 18.875 | 18.927 |
| 0.6 | 18.603 | 18.729 | 18.705 | 18.790 | 18.844 |
| 0.8 | 18.410 | 18.631 | 18.631 | 18.676 | 18.807 |
| 1 | 18.470 | 18.334 | 18.595 | 18.712 | 18.650 |

As you can see there is no consistency with the results, so we will perform another parameter sweep testing a higher deviation to see how it affects the algorithm.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Deviation\Mutation Rate | 0.2 | 0.4 | 0.6 | 0.8 | 1 |
| 1 | 18.24 | 18.56 | 18.57 | 18.69 | 18.78 |
| 2 | 17.45 | 17.73 | 18.07 | 18.04 | 18.15 |
| 3 | 21776.97 | 34894057.65 | 219641917.81 | 2276030145.41 | 5662670362.18 |
| 4 | 121335443.78 | 9069993204.1 | 18215653715.0 | 26889297206.2 | 34902567331.2 |
| 5 | 1720041289.06 | 16608393733.2 | 24922817629.1 | 28276931880.1 | 33455774230.8 |

As we can see from the parameter sweep above, the higher the deviation the further away from a solution the algorithm gets. The reason for this is when each gene is being mutated, it is multiplied by the deviation which produces higher gene value, the deviation affects an individual’s gene similarly to how the mutstep does, instead the deviation will multiple values instead of adding them which has a bigger affect on the gene values hence the huge value changes in the parameter sweep. In figure 6, it shows that having a higher deviation significantly reduces the chances the algorithm will find the better solution. We can also observe that on the left-hand side graph of figure 6, that for some of the iterations the lines are plateauing and then continue to look for new solutions. This down to roulette wheel selection because it will pick a random individual which may not be the best, this means that it could pick a worse individual in a population that does not get the algorithm closer to a solution.

Chart, histogram

Description automatically generatedFigure 5: Mutation rate: 0.2, Deviation: 2 Best Individual: 17.452965558399235

Figure 6: Mutation rate: 0.2, Deviation 4 Best Individual: 121335443.78

3.4 Test 4

For this test we will be looking at Tournament Selection, Single-point crossover and gaussian mutation using the Ackley function. The default parameters we have set are

* Epochs: 500
* Population size: 50
* Gene Size: 20

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Deviation\ Mutation rate | 0.2 | 0.4 | 0.6 | 0.8 | 1 |
| 0.2 | -22.718 | -22.718 | -22.718 | -22.718 | -22.718 |
| 0.4 | -22.718 | -22.718 | -22.718 | -22.718 | -22.718 |
| 0.6 | -22.718 | -22.718 | -22.718 | -22.718 | -22.718 |
| 0.8 | -22.718 | -22.718 | -22.718 | -22.718 | -22.718 |
| 1 | -22.718 | -22.718 | -22.718 | -22.718 | -22.718 |

Chart

Description automatically generated

Figure 5: Mutation rate:0.2, Deviation: 0.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Mutstep\ Mutation Rate | 0.2 | 0.4 | 0.6 | 0.8 | 1 |
| 1 | 8,493,201,333.56 | 8,900,710,264.97 | 6,989,918,066.41 | 5,666,734,604.99 | 6,325,596,689.30 |
| 2 | 6,428,576,798.13 | 4,936,502,771.35 | 5,112,365,704.55 | 4,329,793,654.97 | 3,083,656,384.93 |
| 3 | 4,555,329,243.03 | 3,570,378,787.91 | 2,205,160,392.40 | 2,242,742,435.69 | 1,852,733,651.57 |
| 4 | 3,400,042,250.95 | 2,367,064,545.75 | 1,690,111,504.84 | 1,125,441,563.98 | 733,706,921.43 |
| 5 | 2,365,223,595.83 | 1,968,153,599.19 | 1,913,970,975.40 | 1,594,319,646.02 | 906,850,793.31 |

As you can see from the table above. The Gaussian mutation method is very effective at getting the algorithm to the global optimum. With any of the parameters used in the parameter sweep the algorithm reached the global optimum. It converges very early in the algorithm’s runtime around 10 generations (this was observed on another graph with less generations).

The gaussian mutation function is extremely good at getting both fitness functions close and even at the global optimum. As we saw with the Rosenbrock function, we had to perform 2 parameter sweeps to get close to a global optimum. This shows that the Rosenbrock function requires far more fine tuning compared to the Ackley.

3.5 Test 5

For this test we will be looking at roulette wheel and mutation, while testing multi-point and whole-arithmetic crossover using the Rosenbrock function. We will be comparing each crossover methods with the same mutation rate and mutstep.

The default parameters are:

* Epochs: 500
* Population size: 50
* Gene size: 20
* Multi-point crossover gap size: 3

Testing for multi-point crossover.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Mutstep\ Mutation Rate | 0.2 | 0.4 | 0.6 | 0.8 | 1 |
| 1 | 8,961,029,584.28 | 7,112,255,132.96 | 6,496,177,944.44 | 6,425,520,728.75 | 7,782,929,799.81 |
| 2 | 6,449,131,665.31 | 6,756,652,333.28 | 4,146,514,535.14 | 2,801,558,096.18 | 3,409,126,149.88 |
| 3 | 4,521,473,649.66 | 2,911,081,444.55 | 1,844,521,944.91 | 2,934,007,289.12 | 1,325,344,153.10 |
| 4 | 3,268,866,768.02 | 2,599,480,218.98 | 1,961,306,814.04 | 1,874,278,926.21 | 930,100,059.64 |
| 5 | 2,043,037,767.92 | 1,693,459,438.04 | 2,307,984,096.06 | 1,233,690,746.78 | 471,023,552.03 |

The table above shows that with these parameters, we are still getting nowhere near an optimal solution, however it is showing a pattern. We can see in this case that the higher we have the mutstep the better the individual performs compared to others in the parameter sweep. This is because if we had a higher mutstep which can allow the algorithm to search the problem space quickly.

Testing for whole-arithmetic crossover.

We can see that from both parameters sweeps that we very far from an optimal solution. This is either because the default parameters are not good enough or something else is going on. The new default parameters are

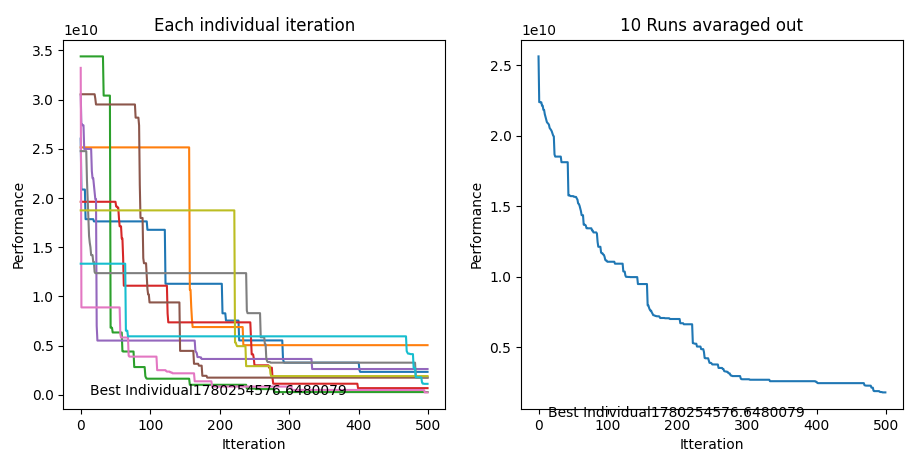
* Epochs: 500
* Population: 150
* Gene size: 20
* Multi-point crossover gap: 6

Chart, histogram

Description automatically generatedwhole-arithmetic crossover testing.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Mutstep\ Mutation rate | 0.2 | 0.4 | 0.6 | 0.8 | 1 |
| 2 | 7,420,756,063.45 | 8,493,569,981.44 | 7,508,766,762.13 | 6,100,782,137.61 | 6,195,861,657.74 |
| 4 | 6,363,566,130.57 | 5,253,936,976.17 | 4,149,201,875.95 | 4,909,517,985.85 | 5,614,713,765.89 |
| 6 | 4,033,956,566.75 | 4,217,074,007.57 | 9,212,554,207.29 | 5,533,005,132.05 | 4,700,346,359.19 |
| 8 | 3,590,212,879.58 | 3,464,413,150.37 | 3,405,316,852.81 | 4,009,282,398.11 | 2,889,941,765.77 |
| 10 | 6,310,596,214.69 | 2,859,134,485.66 | 3,086,076,782.81 | 3,417,142,958.66 | 1,641,932,272.67 |

Figure 6: Mustep: 10 Mutation rate: 1 Best Individual: 1,641,932,272.67

Multi-point crossover testing.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Mutstep\ Mutation Rate | 0.2 | 0.4 | 0.6 | 0.8 | 1 |
| 2 | 6,880,008,747.19 | 7,711,468,439.64 | 9,081,089,051.44 | 6,693,858,698.89 | 6,525,858,618.33 |
| 4 | 6,564,423,460.39 | 6,612,161,776.96 | 7,947,578,717.92 | 4,323,848,539.44 | 5,108,406,696.10 |
| 6 | 4,326,042,981.45 | 5,360,537,612.32 | 7,143,736,393.70 | 3,638,754,794.12 | 3,464,075,524.73 |
| 8 | 4,683,050,827.30 | 4,245,858,871.73 | 5,950,239,542.77 | 4,150,588,238.15 | 2,276,735,304.57 |
| 10 | 4,361,445,028.18 | 2,893,396,607.55 | 3,038,398,917.92 | 2,373,596,838.31 | 1,780,254,576.64 |

Figure 7: Mutstep: 10, Mutation rate: 1, Best individual: 1,780,254,576.64

The second parameter sweep shows is little to no improvement when changing the population size and the gap size for the multi-point crossover. We can see from figure 6 and 7 there is very little difference in how the graph converges and how quickly it is searching. There has to be something else that is affecting the performance this way. We kept all parameters and functions the same, but we swapped out roulette wheel for tournament selection to see what would happen. We can see when comparing figure 6/7 with figure 8 that there is a much quicker convergence, and the algorithm is finding better solutions using tournament selection over roulette wheel selection.

Chart

Description automatically generated

Figure 8: Tournament selection with Mutstep: 10, Mutation rate: 1, Best Individual: 3398150.18

3.6 Test 6

For this test we will be looking at roulette wheel and mutation, while testing multi-point and whole-arithmetic crossover using the Ackley function. We will be comparing each crossover methods with the same mutation rate and mutstep. The new parameter gap size means how many genes could be swapped with another.

The default parameters are:

* Epochs: 500
* Population size: 50
* Gene size: 20
* Multi-point crossover gap size: 3

Here we are testing the multi-point crossover method.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Mutstep\Mutation rate | 0.2 | 0.4 | 0.6 | 0.8 | 1 |
| 1 | -19.80 | -19.29 | -18.84 | -18.51 | -18.47 |
| 2 | -18.87 | -18.01 | -17.18 | -16.97 | -16.34 |
| 3 | -18.38 | -17.29 | -16.32 | -15.41 | -15.15 |
| 4 | -17.44 | -15.75 | -14.96 | -14.30 | -13.75 |
| 5 | -16.60 | -15.18 | -14.30 | -13.23 | -14.51 |

Chart, histogram

Description automatically generated

Figure 9: mutation rate: 0.2, mutstep: 1

Chart, line chart

Description automatically generatedHere we are testing the whole-arithmetic crossover.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Mutstep\ mutation | 0.2 | 0.4 | 0.6 | 0.8 | 1 |
| 1 | -18.66 | -19.81 | -19.64 | -19.60 | -19.47 |
| 2 | -19.68 | -19.22 | -18.78 | -18.35 | -18.00 |
| 3 | -19.24 | -18.45 | -18.02 | -17.44 | -16.91 |
| 4 | -18.89 | -17.87 | -16.92 | -16.35 | -15.85 |
| 5 | -18.63 | -17.32 | -16.39 | -15.49 | -14.80 |

Figure 10: mutation rate: 0.4, Mutstep: 1

Here we can see that having the same parameters for both crossover functions yield the same results. With whole-arithmetic crossover it seems to have a big drop and then steadily decrease. The sudden drop could be down to the roulette wheel selection, this means it could pick a random individual with low or high fitness. In this case, it picked a high fitness individual which caused it to get closer to a solution. For Figure 9, it shows that it converges quicker to a solution than whole-arithmetic crossover, at around 300 generations. In figure 10, we can see with whole-arithmetic crossover it has a slopped convergence compared to figure 9, where it will quickly converge to a solution. Whole arithmetic does this because every gene in each individual is being averaged which causes the graph to steadily decline to a solution rather than the sudden convergence, we have in figure 9.

4 CONCLUSIONS

In conclusion, we found that the Rosenbrock fitness function requires fine tuning of the parameters unlike the Ackley function where it requires far less work to get to get close to a solution. We found that the roulette wheel selection is far worse than tournament selection because of the inherit randomness of the selection process and also the gaussian mutation method was very effective at finding good solutions for each fitness function. To improve on this report in the future we would perform a more in-depth comparison of the different selection, crossover and mutation methods to see how they all stack up against each other to see which methods outperform others.

REFERENCES

Arora, S. and Singh, S., (2019). Butterfly optimization algorithm: a novel approach for global optimization. *Soft Computing*, *23*(3), pp.715-734.

Li, G., Shuang, F., Zhao, P. and Le, C., (2019). An improved butterfly optimization algorithm for engineering design problems using the cross-entropy method. *Symmetry*, *11*(8), p.1049.

Guo, Y., Liu, X. and Chen, L., (2021). Improved butterfly optimisation algorithm based on guiding weight and population restart. *Journal of Experimental & Theoretical Artificial Intelligence*, *33*(1), pp.127-145.