CS7641 - Randomized Optimisation

Student Username: plivesey3
GTID: 90336997

Three different algorithms have been examined for the optimisation problems, including travelling salesperson, flip-flop and continuous peaks. For each of these problems, four optimisation techniques have been applied, including Randomized Hill-Climbing, Simulated Annealing, a Genetic Algorithm and MIMIC. To measure how each of these algorithms performed, a fitness function is applied, which will determine how well the solution performed. The problems are difficult to solve in that they are NP-hard, which means that solutions cannot be found in a reasonable amount of time using conventional methods.

The type of fitness function has a large impact on the results of this process. The choice of function is one way of applying our domain knowledge to the system and so the more reliable and accurate this function is, the more accurate the overall results.

Optimisation Problem Methodology

To solve the three problems, the mlrose Python library was used, which provides algorithms capable of running each of the problems.

Hyper-Parameter Tuning

The following parameters with their given ranges were iterated over to find the optimal selection

For Simulated Annealing:

The Cooling Times: 0.3, 0.5, 0.7, 0.9, 0.99, 0.999.

For Genetic Algorithms:

Mutation-rate: 0.005, 0.05, 0.1, 0.5, and 1.0.
Size of Population: 25%, 50%, 100%, 200%, 300% and 400%.

For MIMIC:

The samples kept for one iteration: 25%, 50%, 60%, 80% and 95%.
The samples taken for one iteration: 25, 50, 75, 100, 125 and 150.

The Algorithms

Randomized Hill-Climbing

This is the most basic form of optimisation examined here. It explores a solution space by starting at a random position in that space and then seeking a local optimum by moving in the direction that increases the solutions overall fitness. Upon finding a local solution, the

algorithm will begin again at another random position in the search space and begin the search again, repeating this until a given number of iterations or until a particular fitness has been achieved

Genetic Algorithms

Genetic algorithms are a form of evolutionary algorithm where new generations are produced by 'mating' other different solutions based on their overall fitness so as to (hopefully) produces more effective solutions.

MIMIC

The final algorithm to be examined is MIMIC. MIMIC tries to make use of the underlying distribution of a problem so has to remember are the parts that were lower in fitness so as to not attempt to find another solution in the same solution space.

Simulated Annealing

Simulated Annealing (SA) works in a way that is similar to Randomised Hill-Climbing, but instead of just moving in a direction that increases the overall fitness, it moves in a way that is kind of simple simulation of annealing used in metallurgy where heating and cooling in a controlled manner is used to align the metal in a way that makes its internal structure stronger. SA holds a simulated temperature that slowly 'cools' over time. When the temperature is high, the solution space will be examined in a way such that the next position may be relatively far away from the current solution, which allows the system to explore the global optimum more effectively. As it cools, the simulation will focus more on more on finding the 'best' local optimum.

The Problems

Travelling Sales Person

The travelling salesperson (TSP) is a problem that asks for the shortest distance to travel between a group of cities, given the distance between each. The solver must begin and end in the same city and each city must be visited once only.

The SA algorithms (figure 1) performed the best for this problem, the highest fitness achieved were by very different decay rates of 0.3 and 0.99 which got to close to -19. The Random Hill Climb performance was extremely surprising, the simplest algorithm achieving the almost highest fitness, varying between 2nd and 3rd position with top fitness of around -20.

The GA (figure 2) performance was less impressive achieving a top fitness between -43 and -42, unsurprisingly with the highest population of 300, although the much lower population of 50 led when the number of iterations was low. Similar things happened with the GA with varying mutation rates (figure 3), the highest fitness being achieved by the highest mutation rate of 1, but the second place taken by the relatively low rate of 0.05. The 0.005 mutation achieved the lowest fitness.

MIMIC was slightly more successful than GA, especially when having a middling keep percentage of 50% which gave a fitness of close to -40. Unsurprisingly the highest

population of 300 gave the best result when varying it, which was similar to the GA population results (figure 4 and 5).

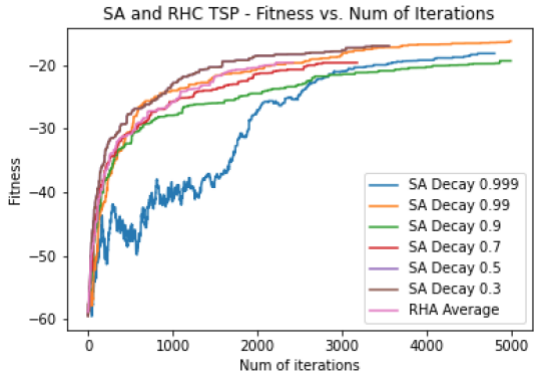


Figure 1

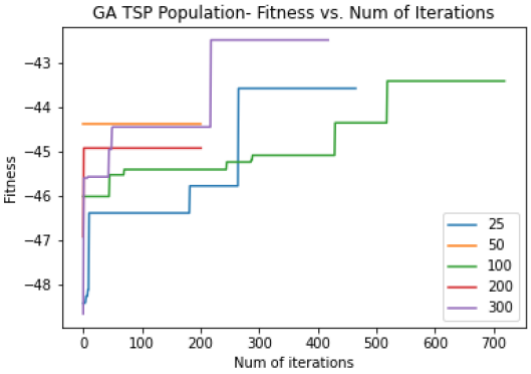


Figure 2

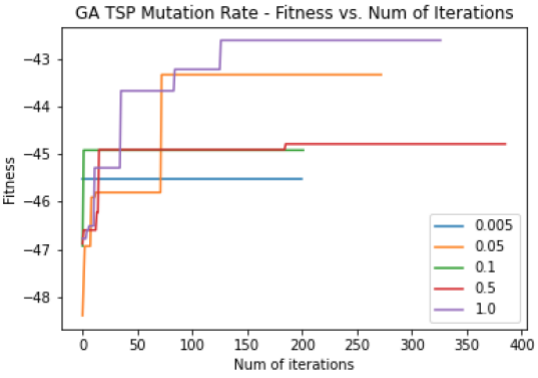


Figure 3

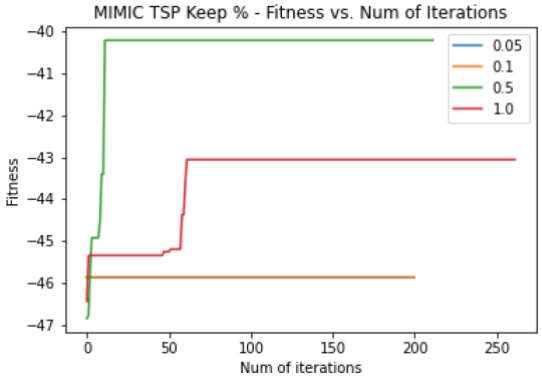


Figure 4

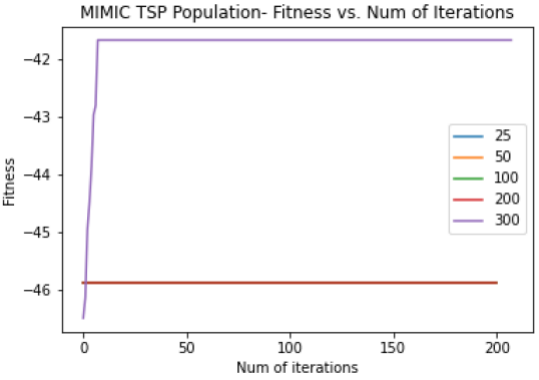


Figure 5

Flip-Flop

For the Flip-Flop problem the solver must attempt to find a a binary bit string in which most of the bits are alternatively set. To understand this, the optimal solution that can be found for strings of a length of 10 would be either '1010101010' or '0101010101', both of which achieve the highest fitness.

The results for the simulated annealing algorithm for the flip flop problem (figure 6) are very similar to those obtained for the TSP. They show that the best results were once again those at opposite decay sizes, with the 0.3 just topping the 0.999 at close to 49. The random hill climbing also did very well once again, mostly in 2nd place for iterations below 500. SA was also once again the better algorithm, although both GA and MIMIC were close.

The genetic algorithm preformed fairly well with the higher 200 and 300 populations once again get the highest fitness of 46. Similar results were also obtained with the mutation rates with a rate of 1.0 and 0.005 achieving the best results once again.

MIMIC achieved slightly higher results compared to GA . Once again, a keep percentage of 0.5 was highest with a fitness at 47 and the bigger population sizes achieved the highest fitness of 47 for the largest population of 300.

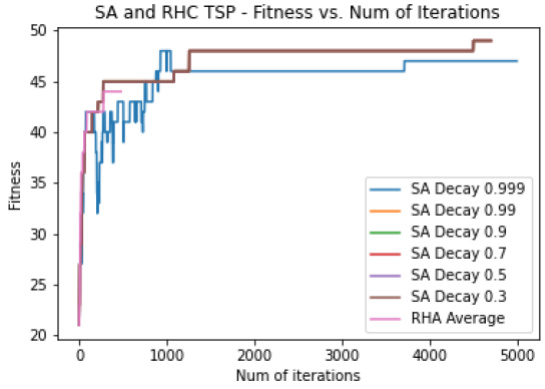


Figure 6

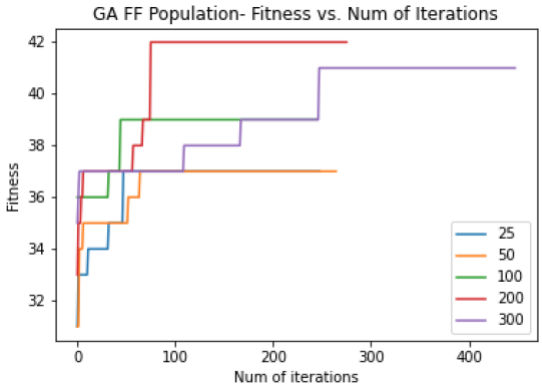


Figure 7

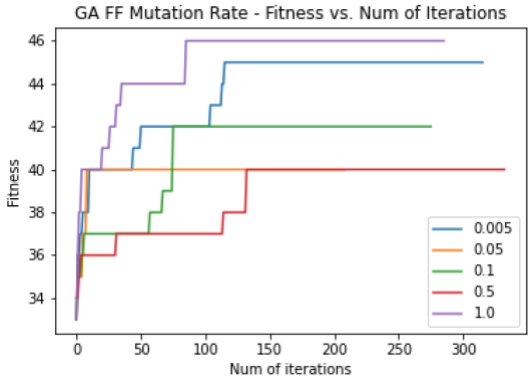


Figure 8

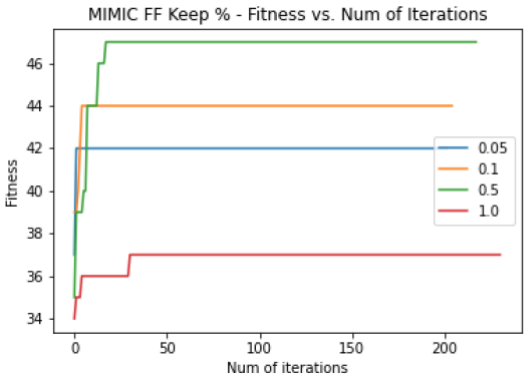


Figure 9

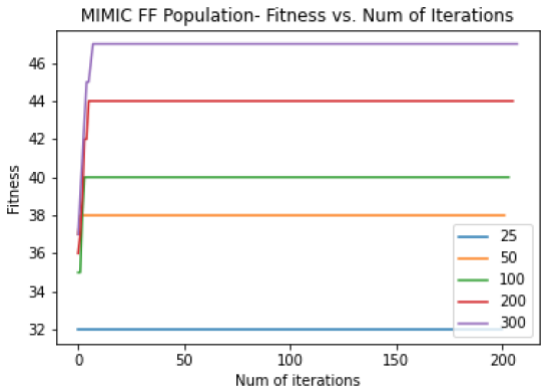


Figure 10

Continuous Peaks

Continuous Peaks (CP) is a problem that evaluates the fitness of an n-dimensional state vector x, given parameter T, as:

$$Fitness(x, T) = \max(max\_run(0, x), max\_run(1, x)) + R(x, T)$$

where:

- $max\_run(b, x)$ is the length of the maximum run of b's in x;
- $R(x, T) = n$, if ($max\_run(0, x) > T$ and $max\_run(1, x) > T$); and
- $R(x, T) = 0$, otherwise. [1]

Once again, similar results are achieved by the SA algorithm, where the highest (0.999) and lowest (0.3) achieved the highest fitness of 37, along with a newcomer, 0.9. The random hill climbing algorithm also does well again for small iterations achieving 36.

With the GA we see something a little different this time with the smaller 50 population winning out at higher iterations with 37, although a variety of different populations achieve (300, 100 and 200 in highest initial peak order) better results on the lower iteration counts. Low mutation rate peak at 36, with the lower rates achieving the peak with lower iterations (0.005 first followed by 0.05, 0.1 and then 0.5).

The MIMICs keep percantage that does best at a fitness of 36 is 0.1, followed by 1.0, 0.5 and then 0.05, which doesn't really seem to follow any order. The best results for the contiuous peaks problem are achieved by MIMIC at a fitness of 37 which is achieved with very few iterations. The populations the achieve this are once again in size order, with a population of 300 achieving the best results (figures 14 and 15).

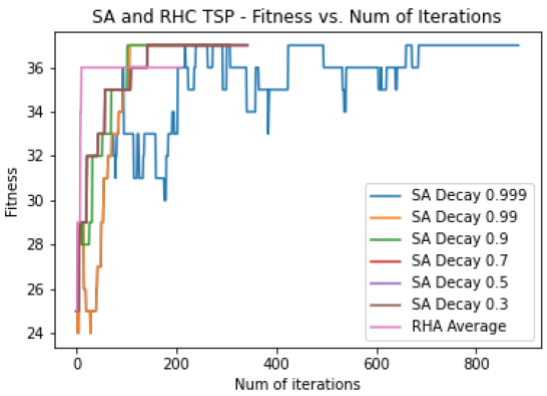


Figure 11

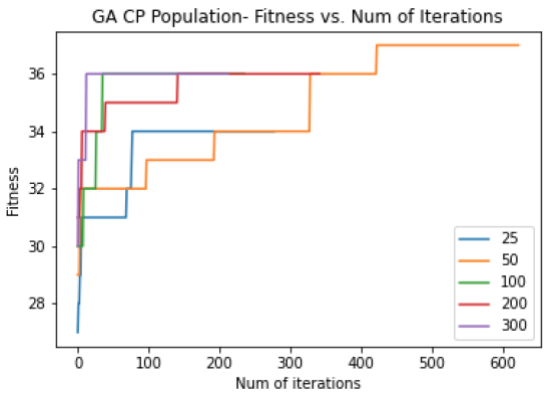


Figure 12

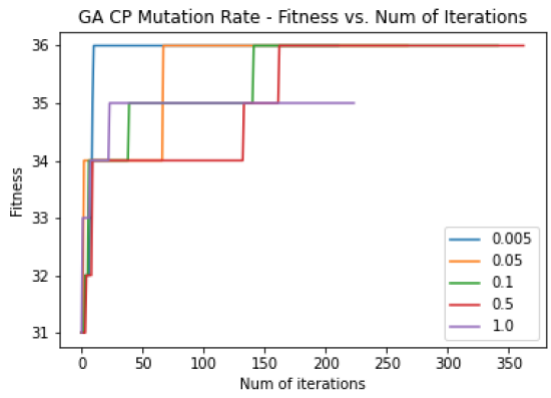


Figure 13

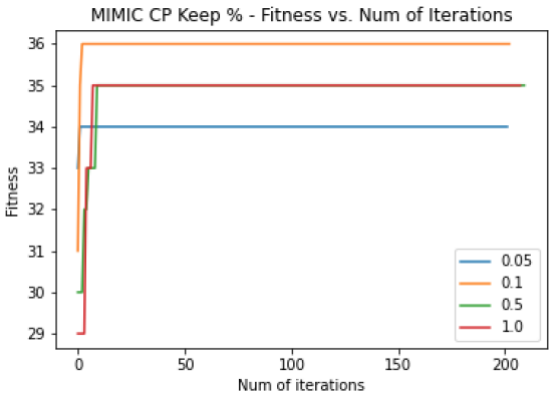


Figure 14

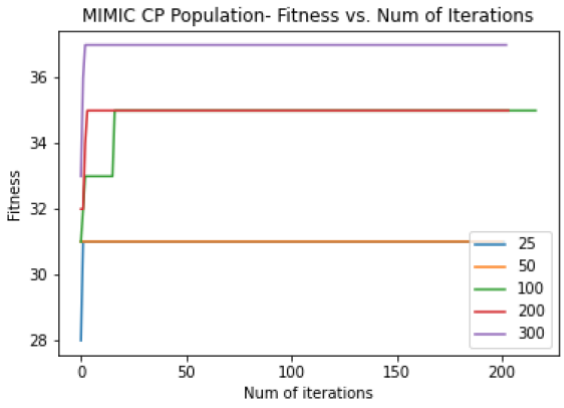


Figure 15

Neural Networks

Three of the algorithms used in the above problems (random hill climb, simulated annealing and genetic algorithms) have been tested with a neural network to optimize the back propagation procedure, along with the more common gradient descent (GD) for comparison. The mlrose python library was again used to create both the randomized optimization algorithms (ROA) and the neural network itself. The dataset from assignment 1 used as a comparison was the Phishing dataset from UCI [2].

The network was tested against each of the ROAs with many combinations of the hyperparameters, but results could not come close to those originally achieved in assignment 1. The f1 score from the 1 assignment was 0.96, whereas here, all that could be achieved was 0.55. The original assignment achieved an very high accuracy of 0.96, whereas here the best that could be achieved was 0.55 again. All of these results were achieved when copying the relevant parameters as closely as possible.

The results that achieved the highest accuracy included a single layer with 25 neurons, a Sigmoid activation function and a learning rate of 0.001. Of these as would be expected due to its popularity, the GD had slightly higher accuracy when training at 0.5529 and joint highest test accuracy at 0.55 (figures 17). Similar results can be seen for the f1-scores

(figures 18) and the precision (figures 19) which almost exactly replicated the accuracy results for all algorithms. The top result for the 3 algorithms we are examining goes to SA at 0.5529 and 0.55 for training and testing accuracy and once again those results were replicated for the f1score and precision.

As with the original problems, the RHC algorithm once again performed admirably achieving results very close to the top 2 with training and testing accuracies at 0.5528 and 0.5491 for training and testing accuracies, the difference being much less than 1% compared with SA and GD. Similar results can be seen for the f1-score and the precision.

Originally, I had once again believed that the genetic algorithm would achieve the highest accuracies, but it proved to be lacking when compared to all of the other by a significant amount getting 0.4514 and 0.4591 for training and testing accuracy and similar for the others.

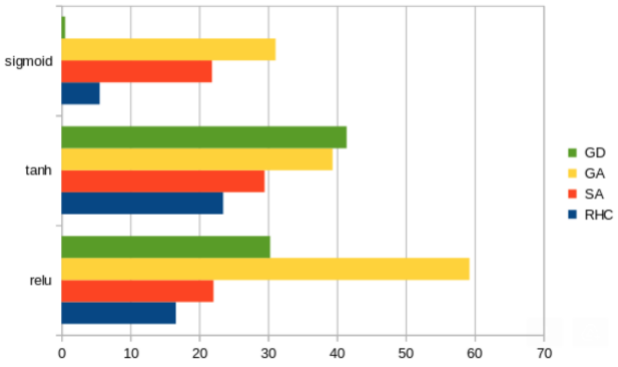


Figure 16 – Algorithm Timings

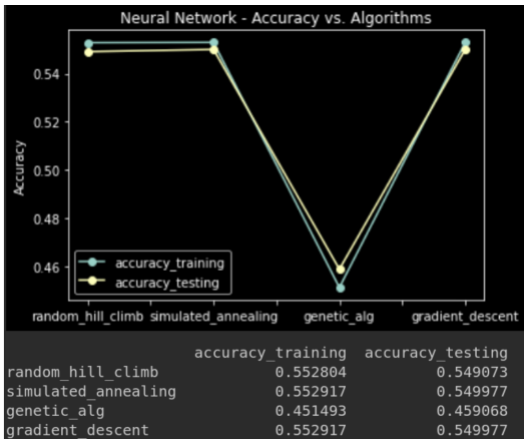


Figure 17 – sigmoid 25 0.001

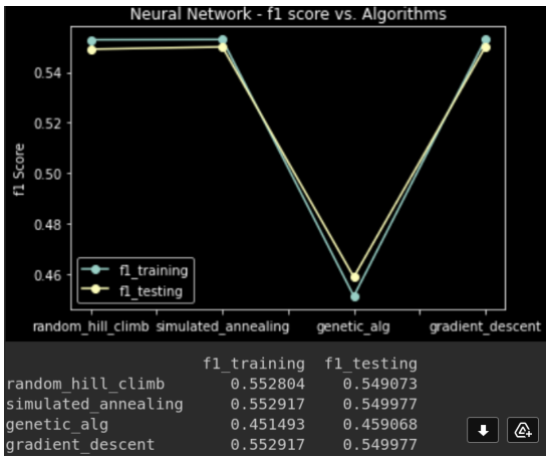


Figure 18 – f1 scores

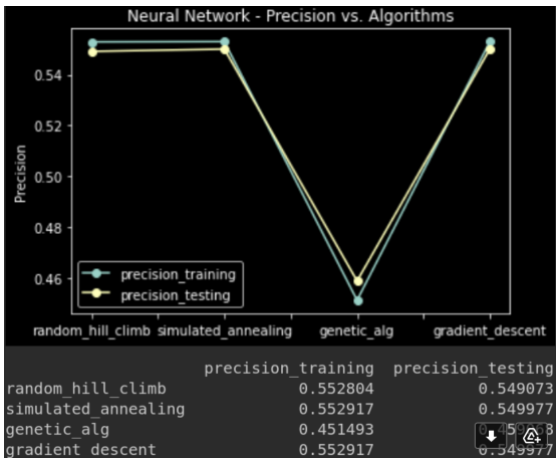


Figure 19 – precision

Comparing three different activation functions, including relu, tanh and sigmoid, there were quite significant differences between each of them. The Sigmoid came out on top as discussed above (0.5529 and 0.55 for SA). Relu and tanh both cam in in second place with both getting 0.5245 from training accuracy and 0.5265 for testing accuracy. (figures 20 and 21).

It is interesting to compare the algorithm timings, both against each other and when comparing the different activation functions. In general, as would be expected, the less

complex RHC was fastest, followed by SA, then GD and finally GA. This fits with expectations when comparing the complexities of the algorithms. One interesting result is that the vary fastest single result was the Sigmoid GD, which was far faster than anything else, coming in at 0.53 seconds.

The overall timings have been reproduced in the following table:

Activation Function	**RHC**	**SA**	**GA**	**GD**
relu	16.62	22.08	59.25	30.31
tanh	23.48	29.5	39.37	41.42
Sigmoid	5.56	21.86	31.09	0.53

Overall, the activation function with highest accuracy, f1 score and precision was the Sigmoid function achieving the fastest times, followed by tanh, with relu coming in last.

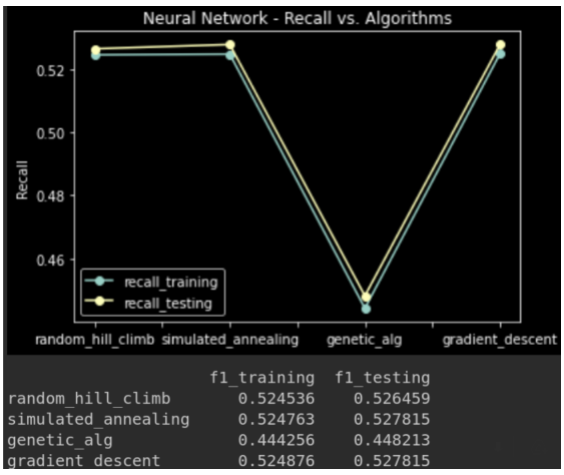


Figure 20 –relu

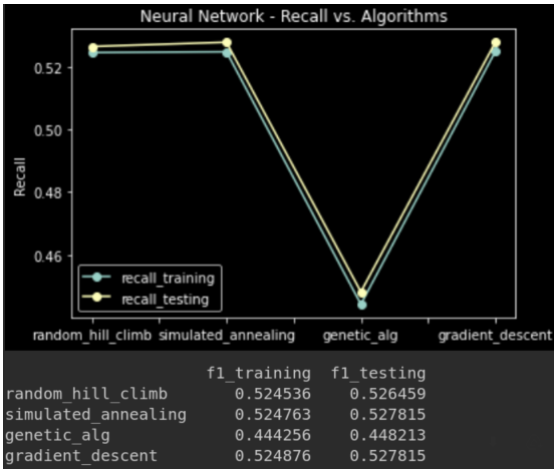
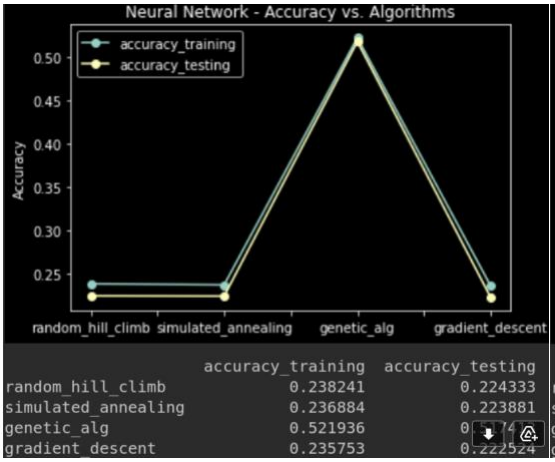


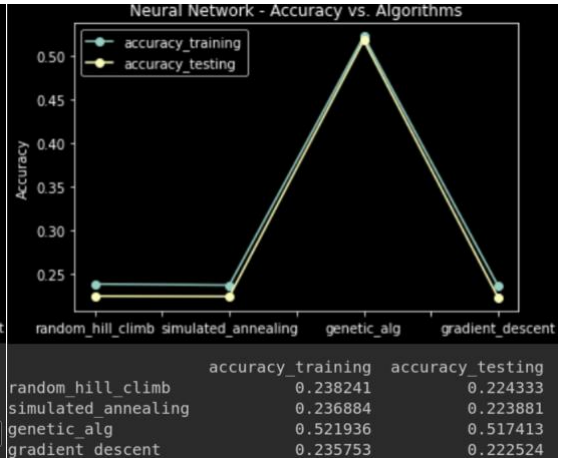
Figure 21 – tanh

Altering the number of neurons had the biggest effect of all of the of the parameter changes. The interesting thing to note here is that although changing the number of neurons by one lower or higher at a count of 24 or 26 significantly decreases the accuracies (by more than one half) for RHC, SA and GD, the GAs accuracy actually increase to 0.5219 for both training and 0.5174 for testing, up close to 10% for both (figure 22 and 23).



	accuracy_training	accuracy_testing
random_hill_climb	0.238241	0.224333
simulated_annealing	0.236884	0.223881
genetic_alg	0.521936	0.517413
gradient_descent	0.235753	0.222524

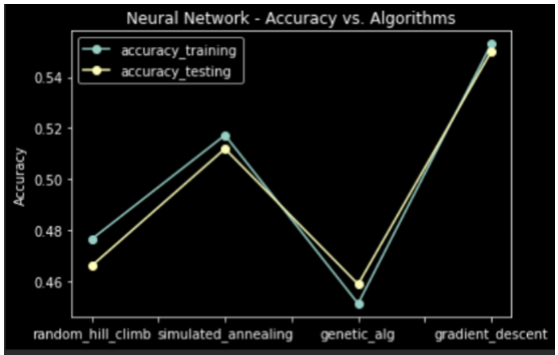
Figure 22 – 24 layer Sigmoid



	accuracy_training	accuracy_testing
random_hill_climb	0.238241	0.224333
simulated_annealing	0.236884	0.223881
genetic_alg	0.521936	0.517413
gradient_descent	0.235753	0.222524

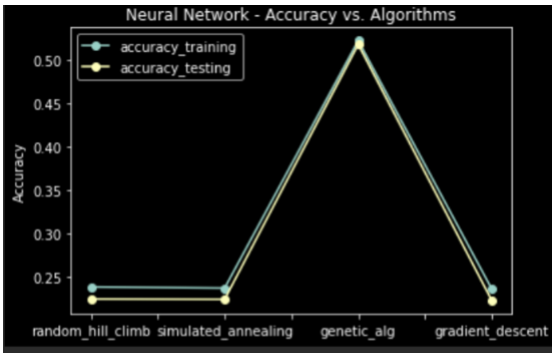
Figure 23 – 26 layer Sigmoid

Finally, altering the learning rate from 0.001 to 0.01 made most of the algorithms less accurate, but gave the exact same result for GD. Similar things happened upon changing the rate to 0.1, but this time it was the GA that improved (figures 24 and 25).



	accuracy_training	accuracy_testing
random_hill_climb	0.476707	0.466305
simulated_annealing	0.517300	0.511986
genetic_alg	0.451493	0.459068
gradient_descent	0.552917	0.549977

Figure 24 – learning rate 0.1



	accuracy_training	accuracy_testing
random_hill_climb	0.238241	0.224333
simulated_annealing	0.236884	0.223881
genetic_alg	0.521936	0.517413
gradient_descent	0.235753	0.222524

Figure 25 – learning rate 0.01

Summary and Improvements

When examining the 3 different random optimisation problems, a number of observations were observed. Firstly, the SA generally performed extremely well, with the lowest and highest decay rates tested (0.3 and 0.999) generally achieving the highest results. It was interesting that the simpler RHC performed very commendably, usually coming very close to the top SA results.

When using the GA algorithm, the higher population counts generally performed better, which is to be expected. When comparing the mutation rates, there was a great deal of variation between the problem sets, although the 1.0 mutation rate generally (surprisingly) achieved the highest fitness.

The MIMIC algorithm showed no real patterns when varying the keep percentage, so no conclusion could be drawn here. Things were different with the population and similar to the GA algorithm, the higher the population, the better the fitness.

There was some variety of outcomes when comparing the different algorithms for the different problems. For the TSP problem, the SA performed significantly better than the others. SA also performed best with the Flip-Flop problem, although MIMIC came a close second. Finally, for the Continuous Peaks problem, MIMIC was best, GA next and the previous winner, SA achieved the lowest fitness.

For the neural network, the GD performed the best with accuracy, precision and f1 scores, but from the three algorithms we were focusing on, SA performed slightly better than RHC in general (matching the optimisation problem results closely), followed a long way behind by the GA.

The Sigmoid activation function achieved the best when compared with relu and tanh as well as being the fastest function. The fastest of all was the GD with the Sigmoid, which was significantly faster than other combinations.

Varying the learning rate and the number of neurons had a lot of impact on the results. This would be a useful place to focus further research. The GA saw significant increases in accuracy when the neurons was altered and the same goes for the GD with the learning rate.

References

[1] Hayes, G. (2019). Fitness functions¶. Retrieved March 14, 2021, from https://mlrose.readthedocs.io/en/stable/source/fitness.html

[2] Mustafa, R., & McCluskey, L. (n.d.). UCI machine Learning repository: Phishing websites data set. Retrieved March 14, 2021, from https://archive.ics.uci.edu/ml/datasets/Phishing+Websites