Randomized Optimization Algorithms

1. RANDOMIZED OPTIMIZATION

Optimization can be defined as the goal of finding the best state/value (which can be a maximum or a minimum) over an input space, with respect to a specific fitness function. The purpose of this project is to explore, analyze, understand and compare the workings of four random search algorithms, using 3 different optimization problems. The second part involves finding optimal weights for a neural network using these algorithms and do a comparative analysis using the dataset from project 1.

2. RANDOM SEARCH ALGORITHMS

2.1. Random Hill Climbing

This algorithm starts at a random point in the input space and by calculating the fitness value of its neighbours, chooses the best value and move towards that space, kind of like trying to climb the hill(in case of maximization) thereby not towards the valleys. When they reach a point, whose neighbours do not have better values, that is returned as the most optimal solution. In random hill climbing, even after reaching a solution, it makes random restarts in the hope of finding the global maxima.

2.2. Simulated Annealing

Simulated annealing algorithm inspired from the chemical process of heating a material and then slowly cooling it. Here, when the temperature is high, we are letting the algorithm explore, randomly wander in the hope that we would avoid local maximums. As we gradually cool, the algorithm focuses on exploiting the input space, i.e. hill climbing and arriving at an optimal solution.

2.3. Genetic Algorithm

Genetic algorithms are inspired from theory of evolution in biology. The algorithm starts out with an initial population, determines the fitness of these individuals and selects the fittest ones as parents. The genes(input values here) are exchanged through crossover, then the off springs go through mutation(random change of values) and they are added to the population. The idea here is arriving at an individual who can no more produce a better fitness, which is the optimal solution.

2.4. MIMIC

The MIMIC algorithm's goal is to model a probability distribution and thereby conveying a structure. We are selecting subsets of inputs and slice the input space by selecting the most probable ones, thereby moving towards the optimum value.

2.5. Algorithm Implementation and Evaluation metrics

I have used the mlrose_hiive package in python. It provides implementations of algorithms and the optimization problems. I have evaluated the algorithms based on their fitness score, running time, number of function evaluations and number of iterations taken to converge.

For simulated annealing, I have used Arithmetic decay of the temperature (cooling) using values [0.001, 0.002, 0.003, 0.004, 0.005].

For genetic algorithm, I have used different mutation probabilities [0.1, 0.2, 0.3, 0.4, 0.5] with a fixed population size of 200.

For MIMIC, I have used different proportions of samples to keep [0.25, 0.50, 0.75] with a fixed population size of 200.

3. OPTIMIZATION PROBLEMS

3.1. Continuous Peaks

Continuous peaks problem is finding the optimal solution for a discrete-valued(0,1) n-dimensional input vector x, maximising the fitness function, where having high number of 0s and 1s yield higher scores. Especially in cases where 0 or 1 counts are greater than a threshold value T, n gets added, thereby boosting the score. Thus, such combinations would lead to lots of local maximums (peaks).

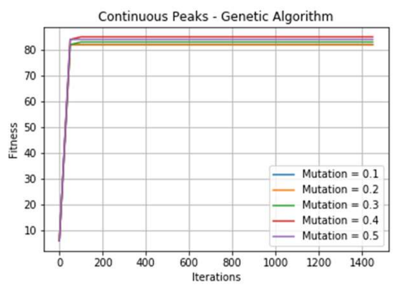
$$Fitness(x,T) = \max(max\_run(0,x), max\_run(1,x)) + R(x,T)$$

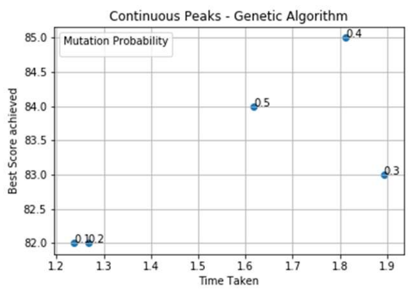
where:

- $max\_run(b,x)$ 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.

Here, I have chosen x as discrete valued(0 or 1) vector of length 50. The threshold T is calculated in the mlrose library as, T = 0.1 * n. Since this problem would have lots of local maximums with wide basins and the attraction basin for the global peak would be narrow, it very interestingly would highlight the advantages of Genetic Algorithm.

Continuous Peaks with Genetic Algorithm





```
Mutation prob:  [0.1, 0.2, 0.3, 0.4, 0.5]
Best scores reached:  [82.0, 82.0, 83.0, 85.0, 84.0]
Time taken to do that:  [1.237412, 1.267158, 1.892486, 1.812498, 1.617163]
Function evaluations taken:  [8052, 9057, 12474, 12478, 10263]
```

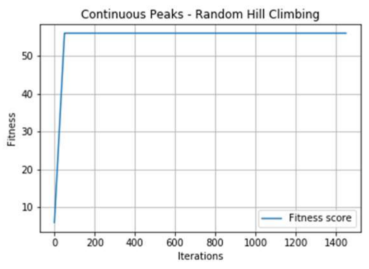
The main advantage of a Genetic algorithm is because of its property of crossover and mutation, it is able to quickly avoid/skip the wide basins of local peaks. Across all the mutation probabilities we have used, the scores are on the high. Genetic algorithms use parallel search across the population domain, which for these kinds of problems play a huge role in eliminating the local maximums and move the search towards the global peak.

At mutation probability 0.4, it yields a high fitness score of 85 in just 1.8 seconds. I would also like to draw attention to the number of iterations it has taken to converge to the best score compared to Simulated Annealing, which highlights the advantages of genetic algorithm. With increased population sizes, we can eliminate even the close local maximums, and approach the global maximum for the continuous peaks problem using the genetic algorithm.

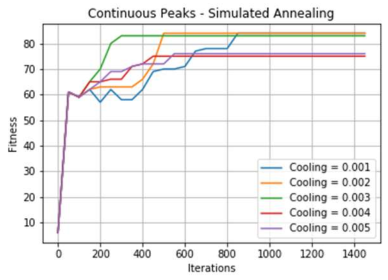
Continuous peaks with RHC and SA

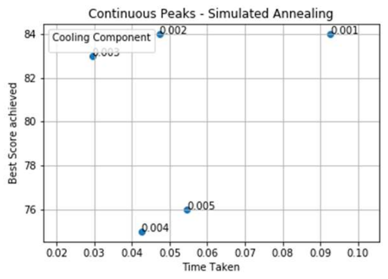
From the results, we can see that RHC has converged at a very low local maximum of 56. It would never be able to reach the global maximum even after multiple random restarts, because there is always a high probability of it falling in the wide basins of local peaks.

Surprisingly, the results for Simulated Annealing are good, but it does take more iterations to converge and only when the cooling is very gradual does it get this score.



```
Best score achieved:  56.0
Time taken to achieve that:  0.001842
Function evaluations taken to achieve that:  13
```

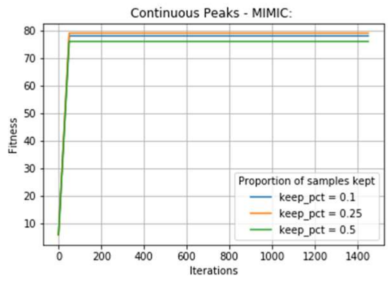


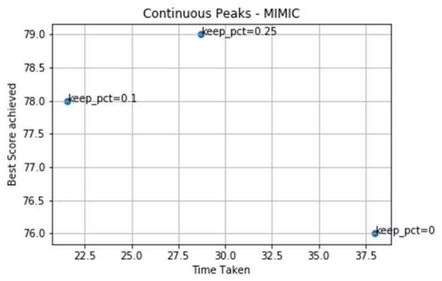


```
decays:  [0.001, 0.002, 0.003, 0.004, 0.005]
Best scores reached:  [84.0, 84.0, 83.0, 75.0, 76.0]
Time taken to do that:  [0.09262, 0.047267, 0.029467, 0.042461, 0.054625]
Function evaluations taken:  [1337, 819, 477, 687, 850]
```

Continuous Peaks with MIMIC

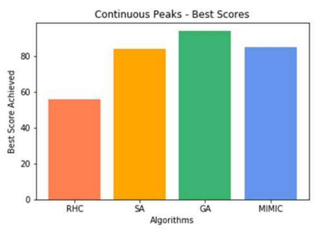
MIMIC has achieved only local maximums but high run time of 28 seconds.

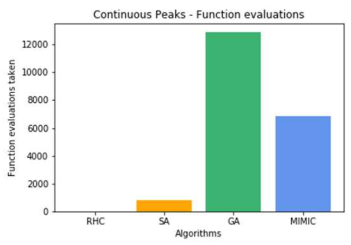


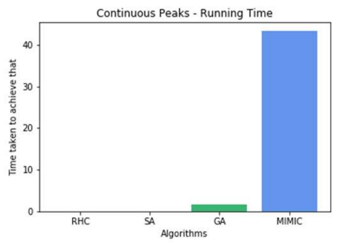


```
Proportion of samples kept:  [0.1, 0.25, 0.5]
Best scores reached:  [78.0, 79.0, 76.0]
Time taken to do that:  [21.58627, 28.719221, 38.006967]
Function evaluations taken:  [3019, 4028, 5231]
```

Comparison of algorithms using Continuous Peaks problem





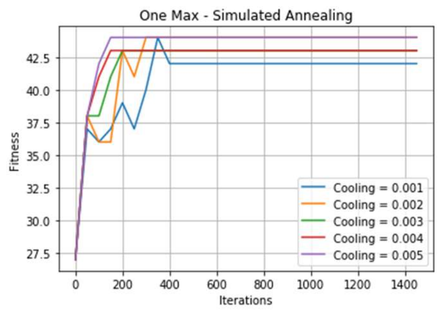


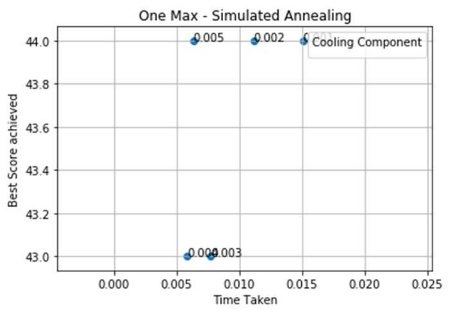
Genetic algorithms has reached the maximum score in less run time in the continuous peak problem.

3.2. One-Max Optimization Problem

One max problem is finding the optimal input vector, which yields the maximum value with respect to the fitness function which is the summation of these values. $Fitness(x) = \sum_{i=0}^{n-1} x_i$, where x is a n-dimensional vector. Here, I have chosen x as discrete valued(it can take either 0 or 1) and a vector of length 50. It is a very simple and elegant problem to capture the strengths and weaknesses of the algorithms. I chose this problem to highlight the advantages of Simulated Annealing.

One-max with Simulated Annealing



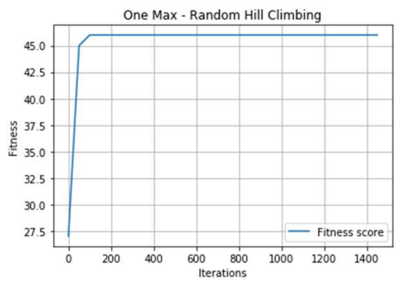


```
decays:  [0.001, 0.002, 0.003, 0.004, 0.005]
Best scores reached:  [44.0, 44.0, 43.0, 43.0, 44.0]
Time taken to do that:  [0.015056, 0.011145, 0.007688, 0.005797, 0.006309]
Function evaluations taken:  [526, 404, 280, 217, 214]
```

From the above graphs we can see that Simulated annealing reached a fitness score of 44 in just 0.006309 seconds and 214 function evaluations.

The main advantage of simulated annealing is how it explores the problem domain before exploiting. From the first graph, its interesting how different cooling values converge at different iterations. The ups and downs unlike in random hill climbing illustrates how the algorithm "explores" when temparture values are high and as it cools it converges to the maximum value. This helps simulated annealing in avoiding the local maximums. From our results, cooling component = 0.005 seems to be the just right amount to yield the best fitness score in lowest run time and least function evaluations. For these kinds of problems, using simulated annealing, tuning and finding just the right cooling component can yield very good scores in very less run time.

One-max with Random hill climbing



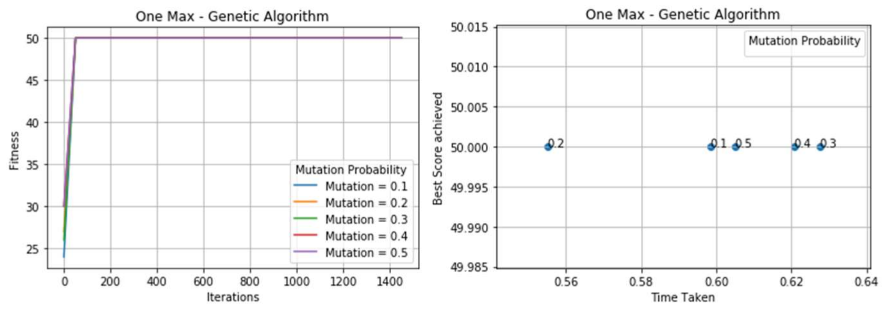
```
Best score achieved:  46.0
Time taken to achieve that:  0.00773
Function evaluations taken to achieve that:  88
```

Random hill climbing has performed good as well. It also largely depends on the random initial state it starts out with. If RHC had started out with an initial state outside the attraction basin, it would be very hard to reach the global maximum. Also, if the input vector size were to be increased and there were more local maximums, it is more prone to getting stuck in lower local maximums.

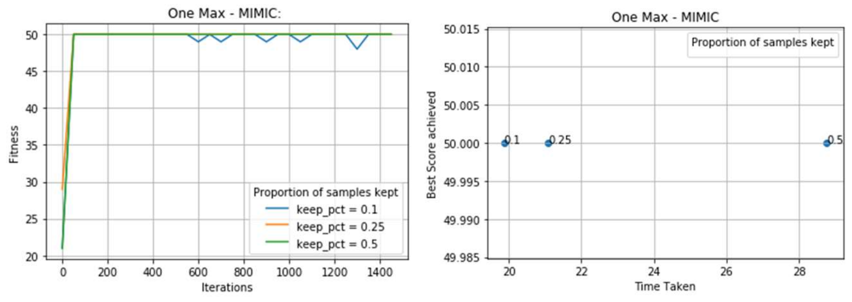
One-max with genetic algorithm & MIMIC

Though genetic algorithm yields the maximum fitness score of 50 for the problem, it takes 0.6 seconds and more than 6000 function evaluations to arrive at it.

The same can be said for MIMIC too, at the cost of 20 seconds which is very high cost of time for these kinds of problems. If the input vector length is increased it is going to be even more hard on the algorithms.



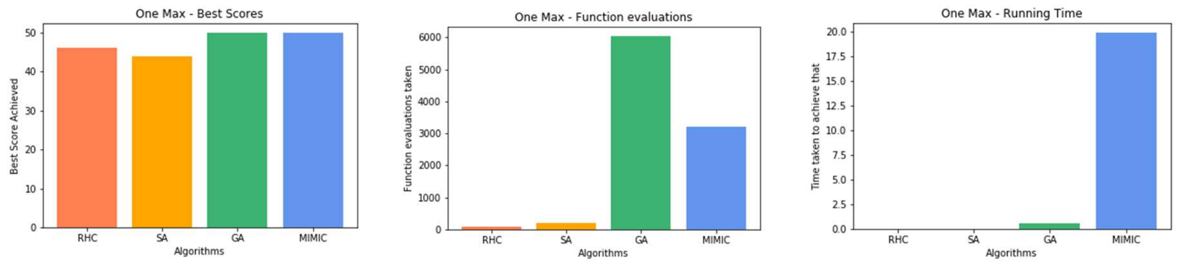
```
Mutation prob:  [0.1, 0.2, 0.3, 0.4, 0.5]
Best scores reached:  [50.0, 50.0, 50.0, 50.0, 50.0]
Time taken to do that:  [0.598413, 0.554985, 0.627527, 0.620722, 0.604987]
Function evaluations taken:  [6440, 6039, 6444, 6242, 6242]
```



```
Proportion of samples kept:  [0.1, 0.25, 0.5]
Best scores reached:  [50.0, 50.0, 50.0]
Time taken to do that:  [19.869137, 21.090407, 28.76488]
Function evaluations taken:  [3221, 3625, 4833]
```

Comparison of Algorithms using One-max problem

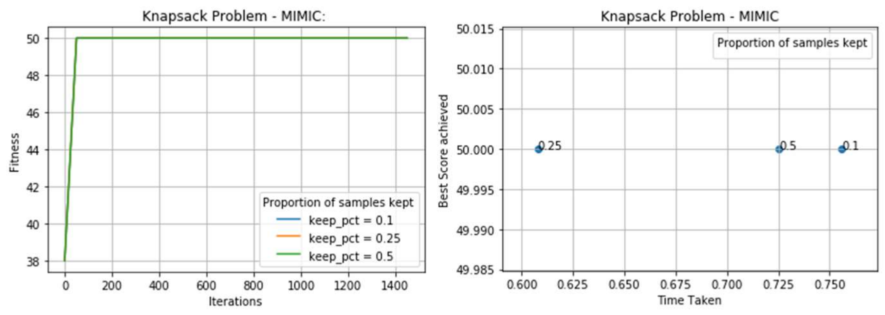
From the bar charts, we can see that Simulated annealing has advantages over genetic and MIMIC in the one-max problem in run time and function evaluations by huge margins.



3.3. Knapsack Optimization Problem

Knapsack problem can be defined as finding what are the optimal objects to place in a knapsack such that we use its maximum capacity. Given a set of weights w, values v for n objects, a maximum weight capacity of the knapsack, over an input space of discrete-valued(0,1) n-dimensional vector x, we need to find the optimal value of x which maximises the total weight with fitness function, $Fitness(x) = \sum_{i=0}^{n-1} v_i x_i, \text{ if} \sum_{i=0}^{n-1} w_i x_i \leq W, \text{ and 0, otherwise,}$. Knapsack is considered a NP-hard problem which can help highlight the advantages of the MIMIC algorithm.

Knapsack problem with MIMIC

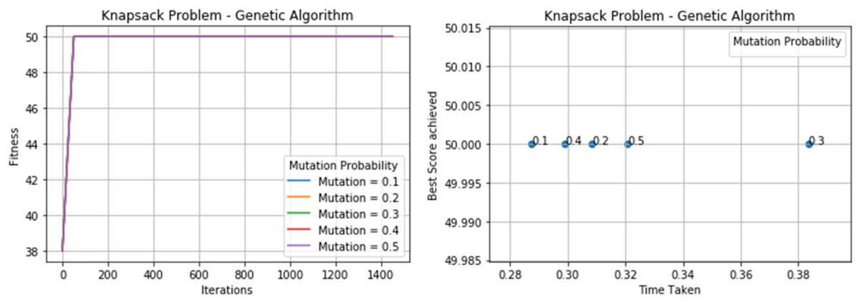


```
Proportion of samples kept:  [0.1, 0.25, 0.5]
Best scores reached:  [50.0, 50.0, 50.0]
Time taken to do that:  [0.755928, 0.608017, 0.725232]
Function evaluations taken:  [2413, 2413, 2615]
```

From the results, we can see that MIMIC achieves high score of 50 evenly across all the values for proportions of samples kept. I want to draw attention to the point that, MIMIC achieves that in very low run time of just 0.6 seconds. If we look at the graphs for performance of MIMIC across the other two problems – one max and continuous peaks which were in the range of 20 seconds, MIMIC seems to be very well suited and most efficient for the Knapsack problem.

MIMIC's advantage of using constructing mutual information matrix and deriving dependencies between features is very helpful at arriving at optimal solution quickly. The important aspect in this problem, is figuring out the dependencies between various input combinations of weights and values. MIMIC retaining probability distributions slices through the input space quickly as it moves towards the global maximum.

Knapsack with Genetic algorithm

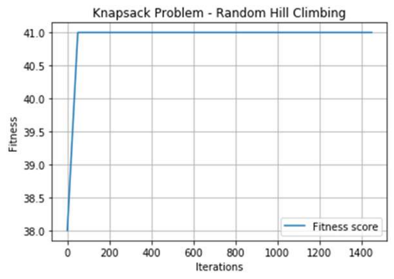


```
Mutation prob:  [0.1, 0.2, 0.3, 0.4, 0.5]
Best scores reached:  [50.0, 50.0, 50.0, 50.0, 50.0]
Time taken to do that:  [0.287459, 0.308467, 0.38353, 0.29904, 0.320778]
Function evaluations taken:  [2615, 2816, 3219, 2615, 2816]
```

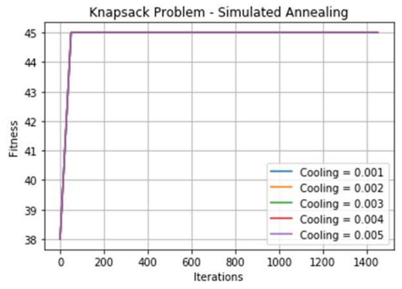
We can see that the genetic algorithm has yielded the high score of 50. I have used 10 objects in the example here. But as the number of objects goes up, MIMIC will be able to outperform genetic algorithms in these problem domains by using its advantage of modeling distributions and thereby creating structures.

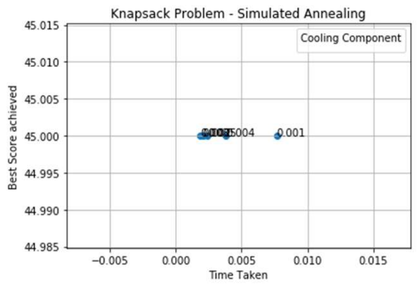
Knapsack problem with RHC & SA

From the graphs below we can see that neither RHC nor SA even across different cooling components get stuck at the local maximums.



```
Best score achieved:  41.0
Time taken to achieve that:  0.002853
Function evaluations taken to achieve that:  18
```

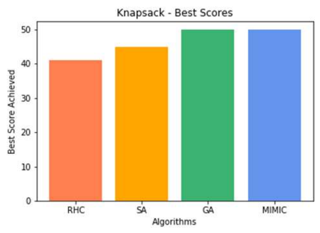


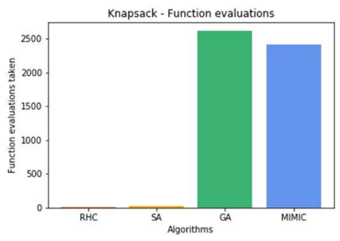


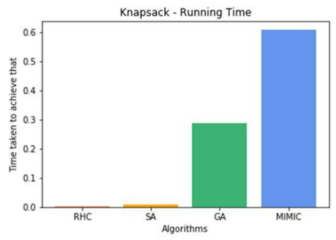
```
decays:  [0.001, 0.002, 0.003, 0.004, 0.005]
Best scores reached:  [45.0, 45.0, 45.0, 45.0, 45.0]
Time taken to do that:  [0.007676, 0.002073, 0.001869, 0.003832, 0.002413]
Function evaluations taken:  [28, 28, 28, 28, 28]
```

Comparison of algorithms using Knapsack Problem

We can conclude that MIMIC is best suited for these kinds of problems in terms of optimal solution, efficient time and function evaluations within very less number of iterations.







4. OVERALL ALGORITHM ANALYSIS

Across the three problem domains, I find Simulated annealing to perform well in terms of run time and function evaluations, though it does not return the best global maximum in few problems, it gets close to other complex algorithms.

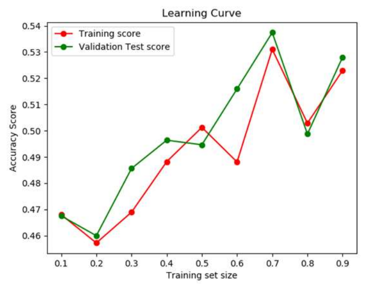
I also found genetic algorithms to perform well in terms of returning high fitness scores. Its second best to simulated annealing in terms of run time. Though they take high number of function evaluations, that's understandable since it employs parallel computing.

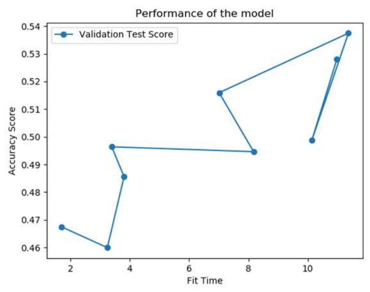
I found MIMIC to perform not well especially in terms of run time, which was very high compared to the other algorithms, and even then could not return best fitness scores. I think this might even have to do with mlrose_hiive's implementation of the algorithm.

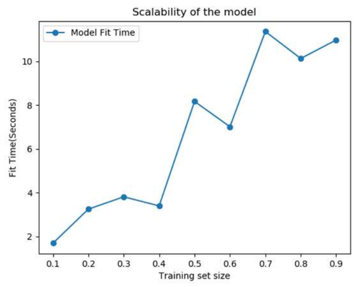
4. NEURAL NETWORKS

I have used the White Wine quality dataset from the UCI ML repository from Assignment 1. In Assignment 1 I tuned the neural network using its hidden layer size parameter. I settled with 3 hidden layers of size 30 which yielded the following accuracy scores.

Graphs from Assignment 1: NN on Wine Dataset



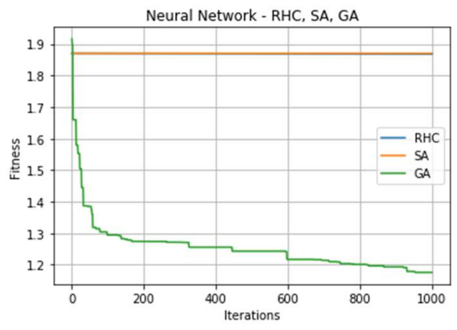




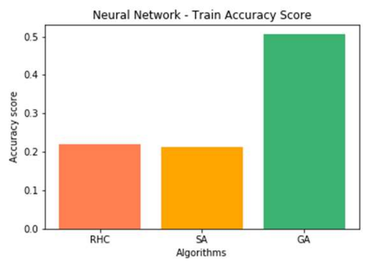
Tuning weights of Neural networks using RO algorithms

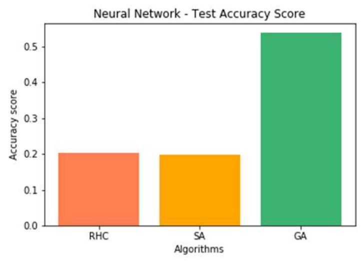
I used the 3 algorithms – RHC, SA, GA to find good weights for the neural network on the wine quality dataset. From the graph below we can see that RHC and SA follow the same line, over the iterations they have fitted on to a state and yield the same score. Fitting weights is out of scope for algorithms such as RHC and SA. They have succumbed to a very low local maximum. It is clear from the low Training and test scores they have yielded after the tuning.

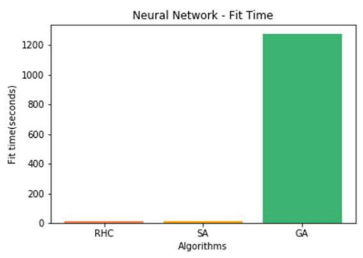
We can see that genetic algorithm has performed way better than RHC and GA, and over iterations it has tuned better weights. This is mainly due to the crossover and mutation properties of the algorithm. They have yielded a training and test score of 0.50 and .53 which are almost the same we found in assignment 1. Their final test score seems to be better than the train score. But they do have a high fit time compared to assignment 1. I suspect this can also be because the genetic algorithm had less default population size and if we can increase that, the fit time would come down and we will be able to boost the accuracy score even better than assignment 1.



```
----------Neural Networks - Random Hill Climbing------------
Training score:  0.22077590607452782
Test Score:  0.20204081632653062
Fit Time:  11.030777
----------Neural Networks - Simulated Annealing------------
Training score:  0.21311893823379274
Test Score:  0.19795918367346937
Fit Time:  12.34732
----------Neural Networks - Genetic Algorithm------------
Training score:  0.5063808065339459
Test Score:  0.5387755102040817
Fit Time:  1274.722984
```







CITATIONS

Hayes, Genevieve. “Fitting a Neural Network Using Randomized Optimization in Python.” *Medium*, Towards Data Science, 24 Dec. 2019, towardsdatascience.com/fitting-a-neural-network-using-randomized-optimization-in-python-71595de4ad2d.

Hayes, Genevieve. “Getting Started with Randomized Optimization in Python.” *Medium*, Towards Data Science, 24 Dec. 2019, towardsdatascience.com/getting-started-with-randomized-optimization-in-python-f7df46babff0.

Mallawaarachchi, Vijini. “Introduction to Genetic Algorithms - Including Example Code.” *Medium*, Towards Data Science, 20 Nov. 2019, towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3.

“Machine Learning, Randomized Optimization and SEarch¶.” *Mlrose*, mlrose.readthedocs.io/en/stable/.

“Knapsack Problem.” *Wikipedia*, Wikimedia Foundation, 22 Feb. 2020, en.wikipedia.org/wiki/Knapsack_problem.