Assignment 2: Randomized Optimization for CS7641 Fall 2021

By Jaeyong Kim

Introduction

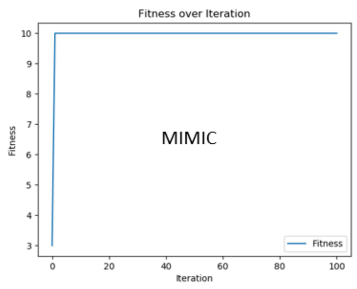
This assignment seeks to evaluate four different randomized optimization to highlight the benefits of each one through its performance on various optimization problems. The four algorithms analyzed are Random Hill Climbing (RHC), Simulated Annealing (SA), Genetic Algorithm (GA), and MIMIC. Initially, these algorithms were applied to discrete problem spaces where they were applied to the following three problems: Four Peaks, One Max, and Knapsack problems. Through exploring various parameters and problem sizes, we can see how each algorithm responds to each problem by evaluating the wall time to optimizing, the fitness over iterations, and the number of function evaluations needed at each iteration. The mlrose-hiive library was used to implement the experiments. [1]

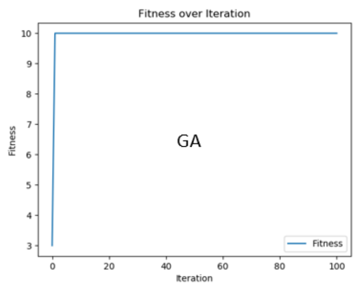
In addition to exploring discrete problem spaces, three of the algorithms were applied to a continuous problem space to optimize the weights in a Neural Network Classifier. Only RHC, SA and GA were used to explore the optimization of weights in the Neural Network Classifier. The White Wine Quality dataset from Assignment 1 was used to analyze the performance of the Neural Network Classifier after going through optimization. [2] Similar to assignment 1, this dataset was split into a binary classification problem by making 'Wine Quality' > 7 as a 1 for good wine and anything less as a 0 for bad wine.

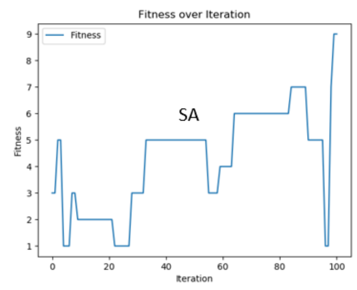
Random Optimization Algorithms

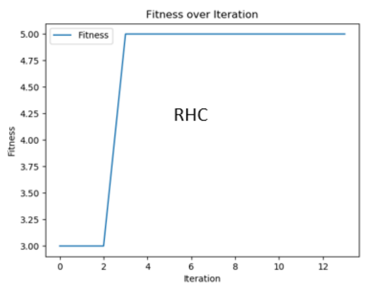
1. Four Peaks

Four Peaks is a problem that includes a space where there exists both local optima as well as global optima. This kind of problem is tricker for algorithms such as RHC and SA because they are more likely get stuck in a region with a local optimum. This is especially the case for RHC especially when there are few to none random restarts available for the algorithm. Due to the greedy nature of these algorithms, they did not perform as well as GA or MIMIC as shown by the charts below.

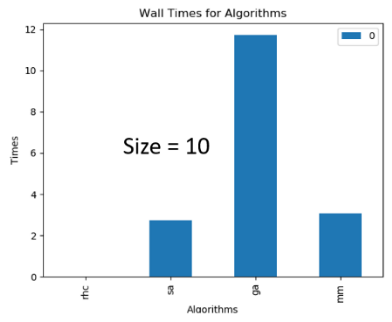


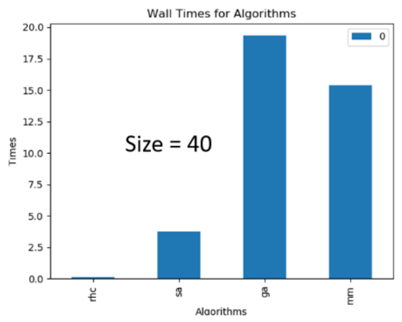


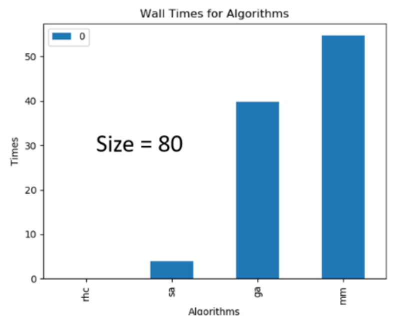


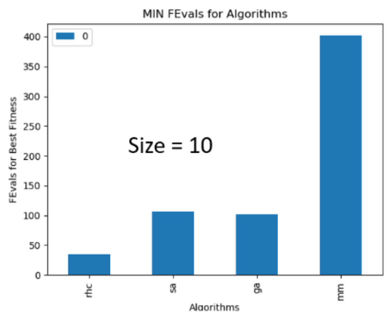


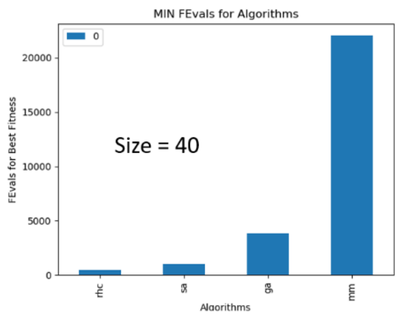
To show how MIMIC excels in this problem, I used a smaller problem space rather than a larger one. In most cases, I found that GA reached higher levels of fitness than other algorithms in much larger problem spaces. In the fitness charts, you see that SA and RHC do not reach the optimal level of fitness which likely means that they are both getting caught in the local optima. The best performers are both MIMIC and GA in terms of Fitness, but wall time shows that MIMIC happens to take only 1/3 of the time that it takes GA. MIMIC can be a faster algorithm because it attempts to use the evaluations from previous iterations in later iterations as it continuously generates a new probability distribution. MIMIC does however seem to be more costly than GA in all sizes of the problem.

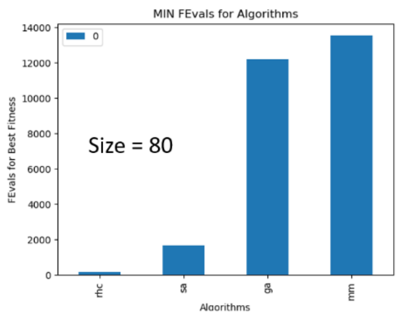








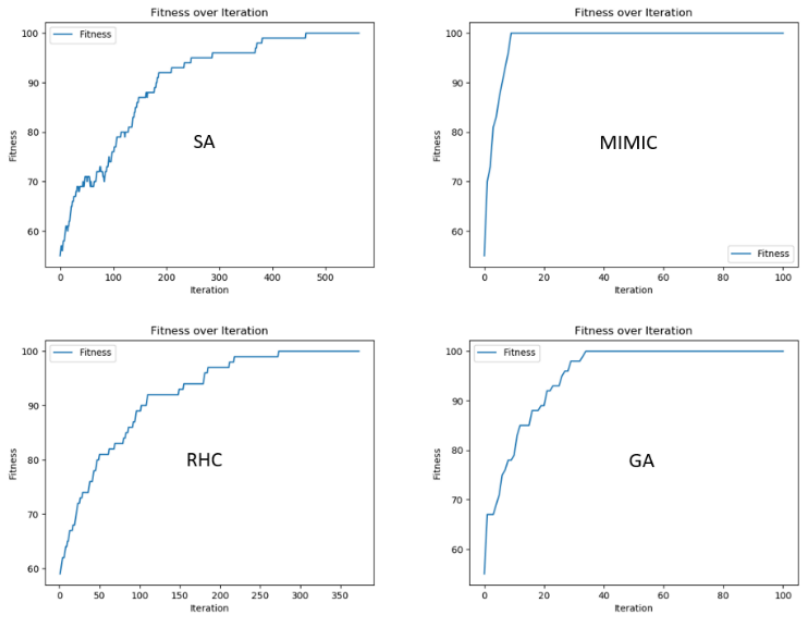


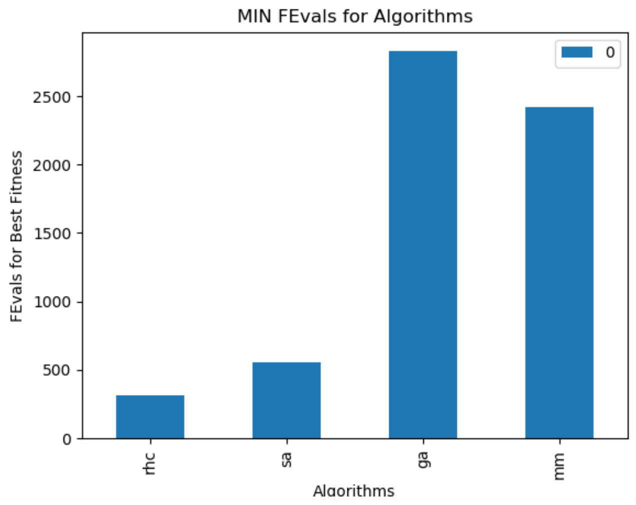


As the four peaks problem grows, GA tends to perform better in terms of Wall Times and Function Evaluations. The progression is shown above.

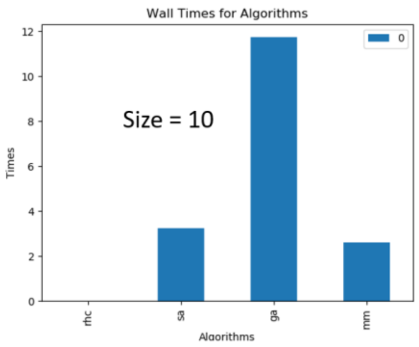
2. One Max

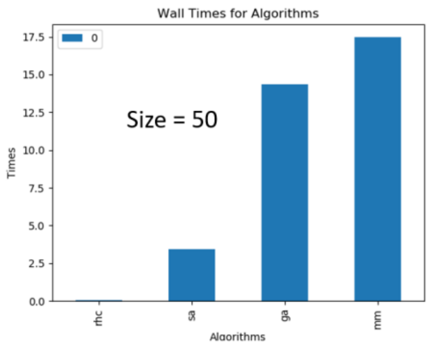
The one max problem involves maximizing the number of ones in an array that can only contain binary values of 0 and 1. It's a relatively simple problem because in a given array of length N, the only global solution is having N ones in that array. This problem is designed to show the positives of utilizing simulated annealing as an optimization algorithm.

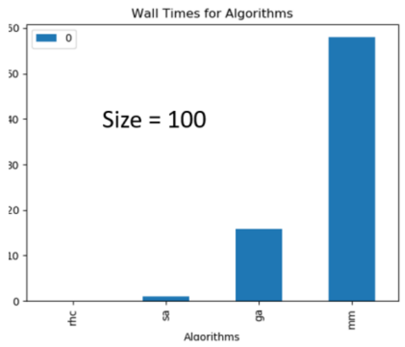




One of the major highlights of simulated annealing is that it is one of the least computationally expensive algorithms that can be utilized. While it may not perform well when there are many local optima, in the case of a single global optima, it produces the maximum fitness very quickly. This is also observed in RHC as the optimization algorithm. The data above was taken from a problem space taking an array of length 100. This computational quickness can also be shown in the graph above showing the number of function evaluations over each iteration, compared to algorithms like GA or MIMIC.



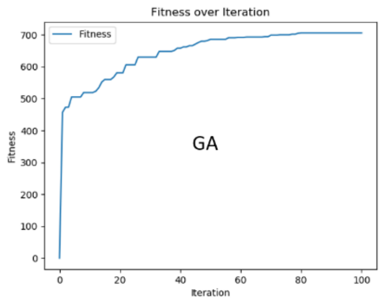


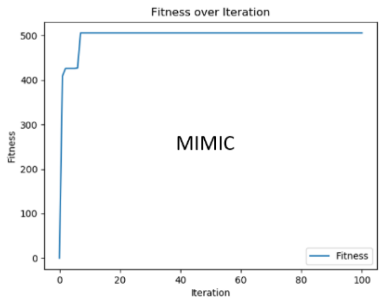


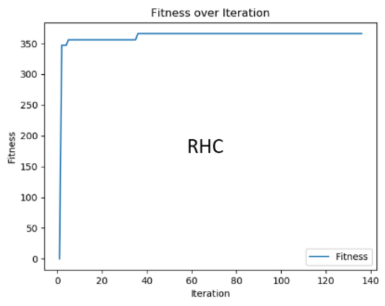
Not only is the wall time faster for SA, but it also took less cost function evaluations to get to that level of fitness. While both MIMIC and GA take less iterations to get to the level of fitness, they both require much larger quantities of function evaluations at each iterative step. It's interesting to see that at smaller problem sizes like arrays of length 10, MIMIC actually performs better than GA, but as the problem space increases, MIMIC becomes more costly and takes longer. This is shown by the figures above.

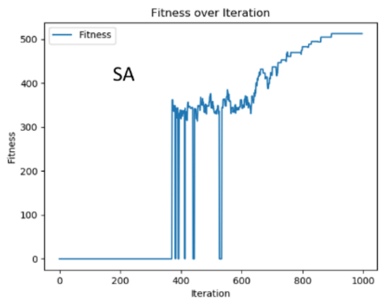
3. Knapsack

The knapsack problem as defined as an optimization problem where there is a knapsack with a total weight and a list of items that each have a specific weight and value. The goal is to maximize the value of items in the knapsack without exceeding the weight. After running through the four algorithms, the GA algorithm performed the best. The charts below show the performance of the different algorithms with Fitness over Iterations as well as Wall Time. They were gathered from analyzing a problem size of 100.

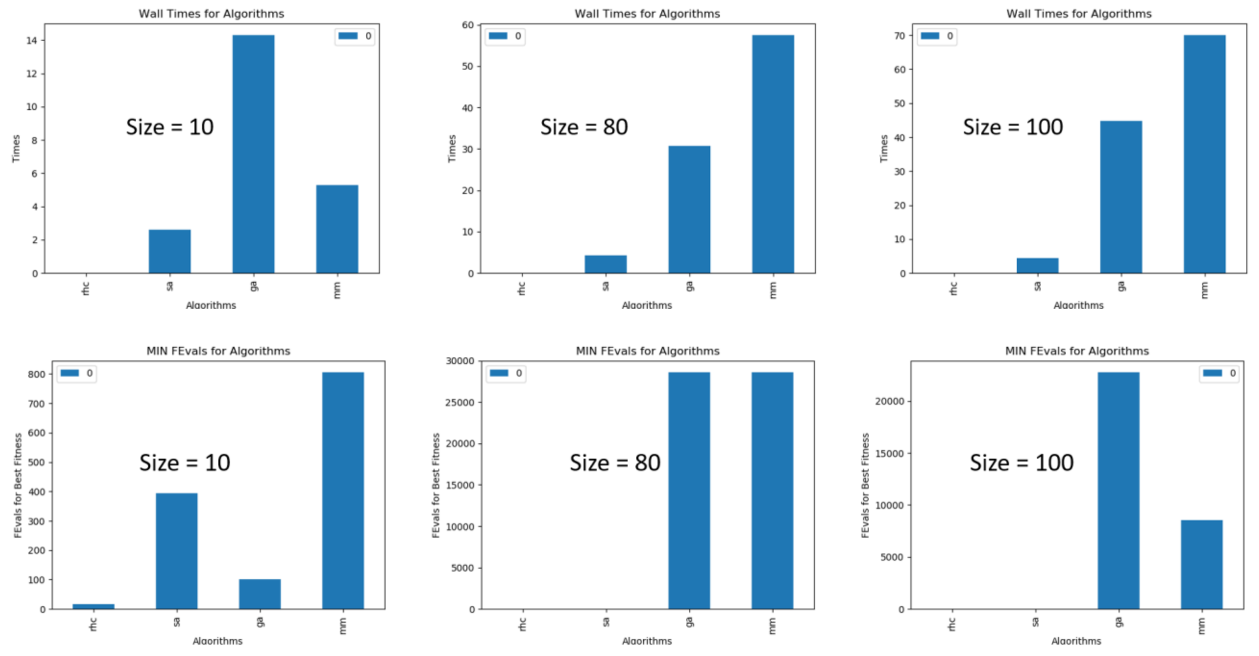








The genetic algorithm reaches the highest fitness out of the four algorithms. In addition to reaching the highest fitness, it also has a relatively quicker performance than MIMIC. From the charts above, it seems like RHC, SA and MIMIC all get stuck at suboptimal fitness levels. Because of how greedy the algorithms are, RHC and SA show poor results in terms of fitness. The mutations that occur in GA show their value in GA as it does not get stuck at local optima like RHC, MIMIC and SA.



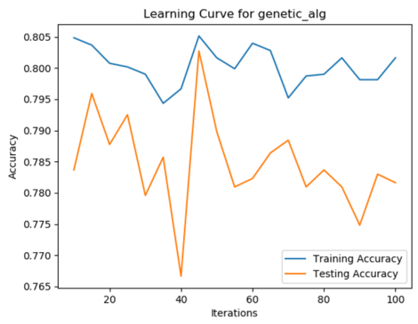
With Knapsack, there is a similar pattern to Four Peaks where GA gets out performed by MIMIC in smaller problem sizes. In the case where we look at a problem size of 100, we see GA being the quicker algorithm compared to MIMIC.

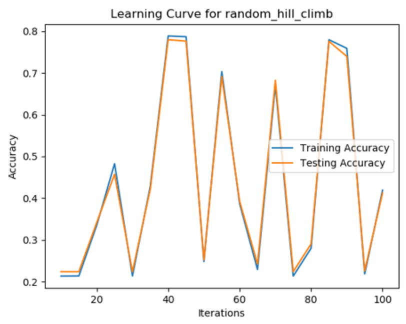
Optimizing Neural Network

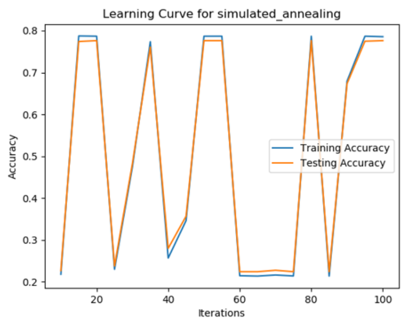
This part of the assignment involved looking at optimizing a Neural Network Classifier that was previously used on a dataset from the first assignment. This classifier was put through 3 of the Random Optimizers above to get the weights for the Neural Network. RHC, SA and GA were evaluated since these support optimizations in a continuous space rather than a discrete space. RHC was utilized with zero restarts to see how it would perform given its greedy nature. For GA, a population size of 250 and a mutation probability of 0.25 was used. These three models were subject to many iterations of training and compared to results of Gradient Descent Optimization. The table below highlights some high level metrics like accuracy, wall time and function evaluations for the optimization algorithms.

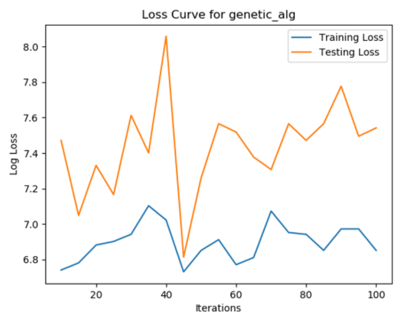
	Training Accuracy	**Testing Accuracy**	**Validation Accuracy**	**Wall Time**	**Function Evaluations**
Gradient Descent	0.46	0.46	0.46	2.57 s	-
Random Hill	0.76	0.74	0.76	0.89 s	358
Simulated Annealing	0.23	0.21	0.23	2.40 s	962
Genetic Algorithm	0.78	0.80	0.78	9.21 s	4774

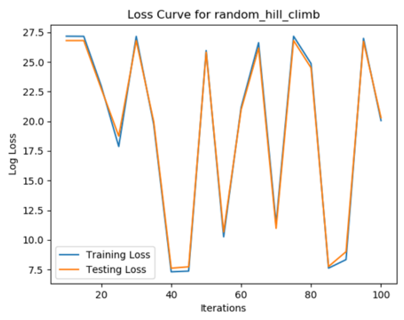
At first glance, it seems as though both Random Hill and Genetic Algorithm performed the best while attempting to optimize the Neural Net classifier to predict labels for the wine dataset. However, after running the optimization over various iterations, it was obvious that GA performed the best over all the iterations. The charts below show the performance over many iterations.

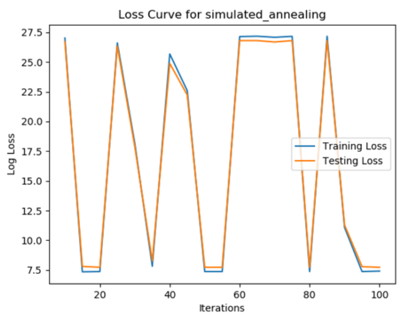




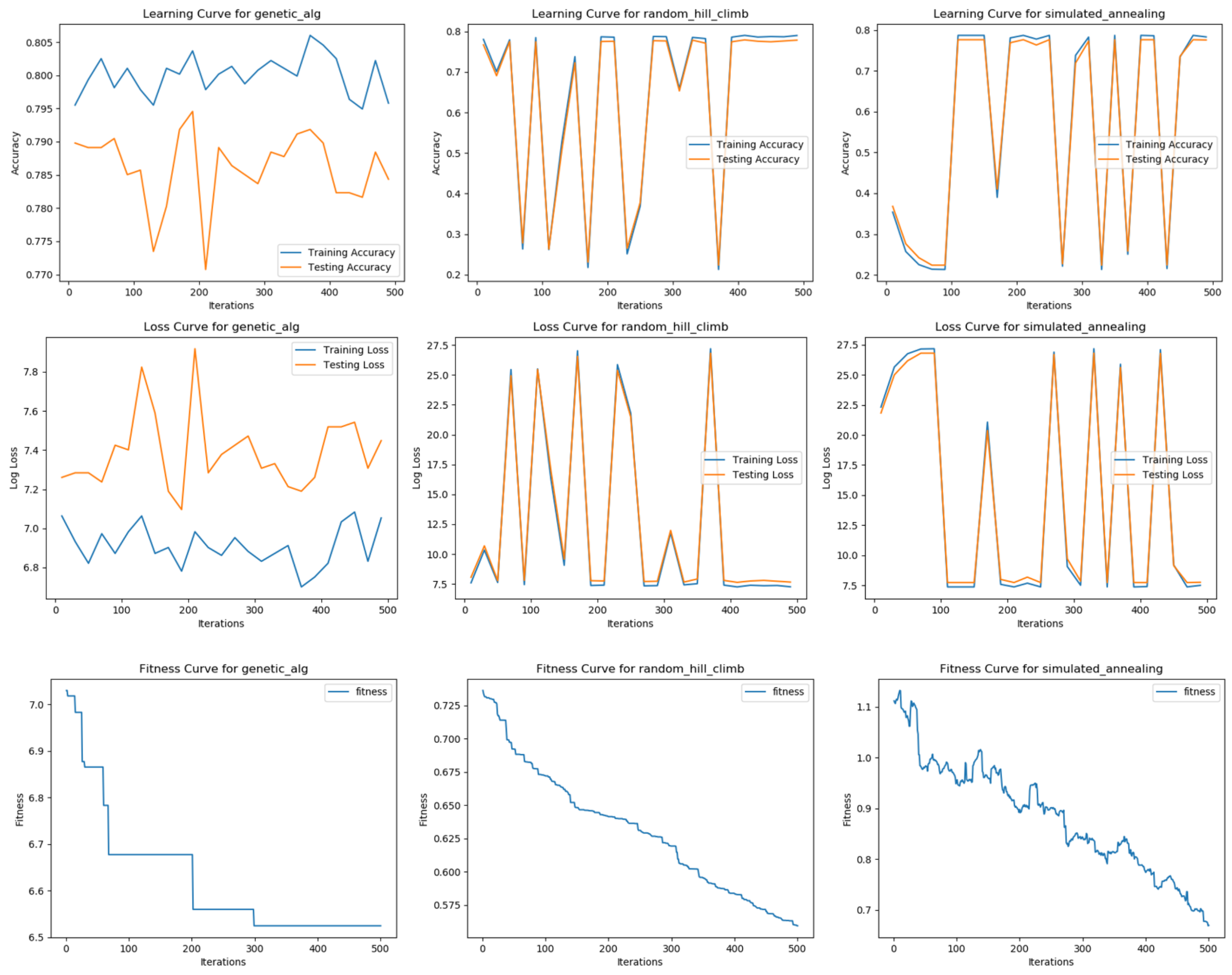








While none of the learning curves or loss curves above seem to show a lot of promise while tuning the weights, we can see that GA tends to work better in terms of optimizing. The reason being is that the range of both the accuracy and loss is much less with GA than it is with RHC or SA. RHC and SA do not consistently provide the same accuracy and loss metrics through each iteration. As the optimizers work on tuning the weights to the NN Classifier, it seems as though RHC and SA have more difficult time finding the optimum point of fitness. We see that for both RHC and SA, the log loss curves will fluctuate from 27.5 to 7.5 and there is no decreases in loss as the iterations continue.



The chart above shows the optimizations run for around 500 iterations and what we see is the same pattern as the charts above. GA us able to hit higher accuracies (or lower loss) with optimizing the weights which seems to indicate that this optimization problem in the continuous space may have some local optima and large basins of attraction which would tend to hinder SA or RHC's ability to find the global optima.

Conclusion

Each algorithm has its own positives and downfalls depending on the type of problem that it tries to solve. Algorithms like Random Hill Climbing and Situated Annealing can be great options where the optimization problem does not have local optima or has distinct a distinct global optima. The low computational cost of these algorithms can be beneficial in many situations. For harder optimization problems that may be hard to find a global optima, shown by Four Peaks or Knapsack, Genetic Algorithm or MIMIC can be better depending on the size of the problem. Even in terms of optimizing the weights of a neural network classifier, GA performed better than both RHC and SA in most instances.

References

1. Hayes, G. (2019). *mlrose: Machine Learning, Randomized Optimization and SEarch package for Python*. https://github.com/gkhayes/mlrose. Accessed: *13 October 2021*.
2. UCI machine Learning Repository: Wine quality data set. (n.d.). Retrieved September 26, 2021, from https://archive.ics.uci.edu/ml/datasets/wine+quality.