

Assignment 2 (Randomized Optimization)

Part 1: Comparison of Randomized Optimization Algorithms

We shall use the following 3 Optimization problems and apply the Randomized Optimization algorithms on them to showcase hyper-parameter tuning and compare the strength and weaknesses.

- Four Peaks
- Continuous Peaks
- Knapsack

Four Peaks:

The 4 Peaks problem was formulated by Baluja and Caruana and designed to be GA friendly.

Given an N-dimensional input vector \vec{X} , the four peaks evaluation function is defined as:

$$f(\vec{X}, T) = \max[\text{tail}(0, \vec{X}), \text{head}(1, \vec{X})] + R(\vec{X}, T)$$

where

$$\begin{aligned} \text{tail}(b, \vec{X}) &= \text{number of trailing } b\text{'s in } \vec{X} \\ \text{head}(b, \vec{X}) &= \text{number of leading } b\text{'s in } \vec{X} \\ R(\vec{X}, T) &= \begin{cases} N & \text{if } \text{tail}(0, \vec{X}) > T \text{ and } \text{head}(1, \vec{X}) > T \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

There are two global maxima for this function.

- when there are T + 1 leading 1's followed by all 0's or
- when there are T + 1 trailing 0's preceded by all 1's.

There are also two suboptimal local maxima.

- string of all 1's or
- string of all 0's.

The above description of the Four Peaks problem is referred from the paper [1]

Problem configuration:

Here we shall consider a bit string of length (i.e., N) = 100 and T as 10% of the bit length. Therefore, the MLRose parameter t_pct is set to 0.1 (i.e., 10 percent on N).

At the global maxima, the fitness value shall be calculated as

$$2N - (T + 1) = 2 \cdot 100 - (0.1 \cdot 100 + 1) = 200 - (10 + 1) = 200 - 11 = 189$$

Where $T = t_pct \cdot N = 0.1 \cdot 100 = 10$

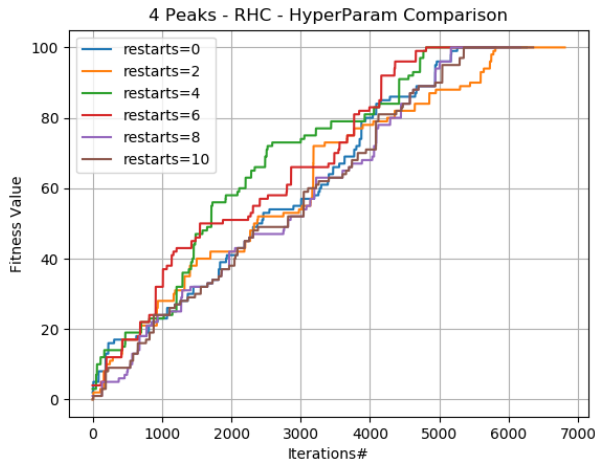
The fitness value at the two local optima is 100. This is independent of the parameter T.

The below graphs represent the Fitness Value achieved the four RO algorithms when ran on varying hyper-param values. We shall select those hyper-param values that reached highest Fitness Value at lower Iterations.

RHC, even with 10 restarts could not cross the barrier of the sub-optimal local optima (i.e., fitness value 100). This is due to the fact that with high value of bit length, the value of T also increases. The size of the basin of attraction around the local optima grows exponentially as T increases where RHC can get stuck.

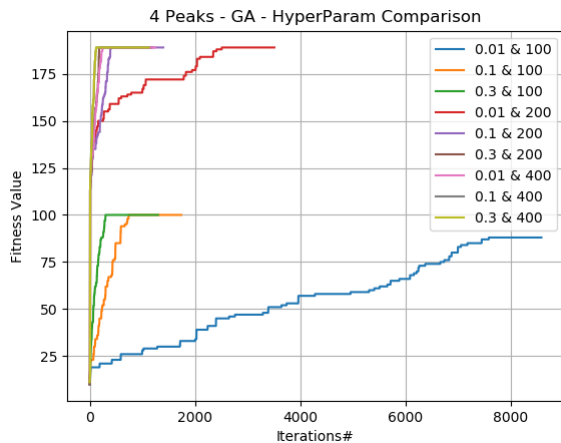
Simulated Annealing with low initial temperature performed similar to RHC and got stuck in the local optima due to the same reason as RHC (i.e., large basin of attraction around local optima). But since SA combines the power of Exploration & Exploitation; starting with a high temperature value (here 10) caused more random movements in the search space (i.e., more exploration). Eventually we

Optimal param value: restarts=4 reaches the local optimal fitness value (i.e., 100) with least number of iterations.



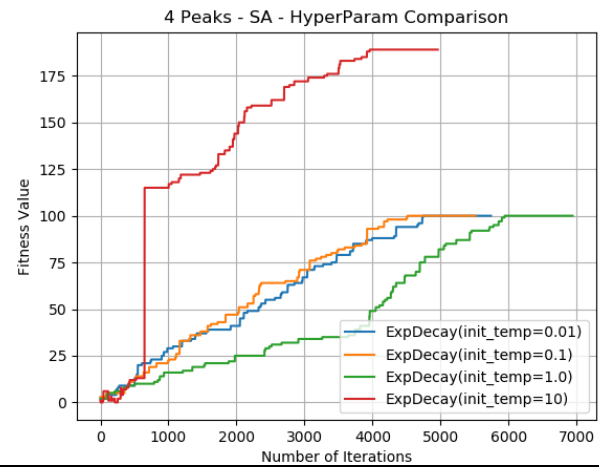
For **Genetic algorithm**, we varied the 'mutation_prob' and 'pop_size' with 9 different combinations. It is interesting to observe that large population size performed better.

Optimal param value: mutation_prob=0.3 & pop_size=400 since it reached the global optimal fitness value at shortest iteration.



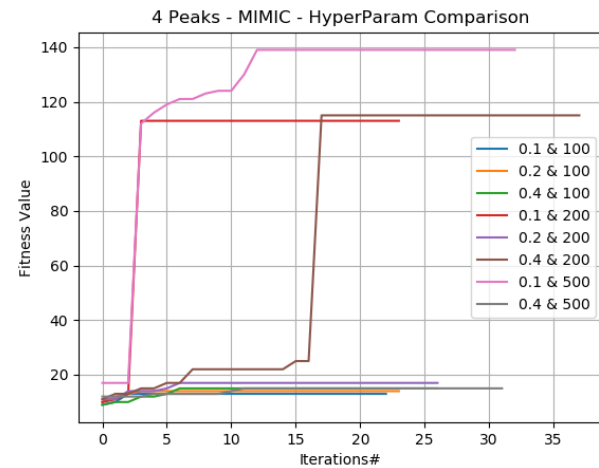
can see that SA was able to reach the Global optimal fitness value of 189 (red line).

Optimal param value: init_temp=10



For **MIMIC**, 'keep_pct' and 'pop_size' was varied for eight different combinations. It looks like larger population size performed better.

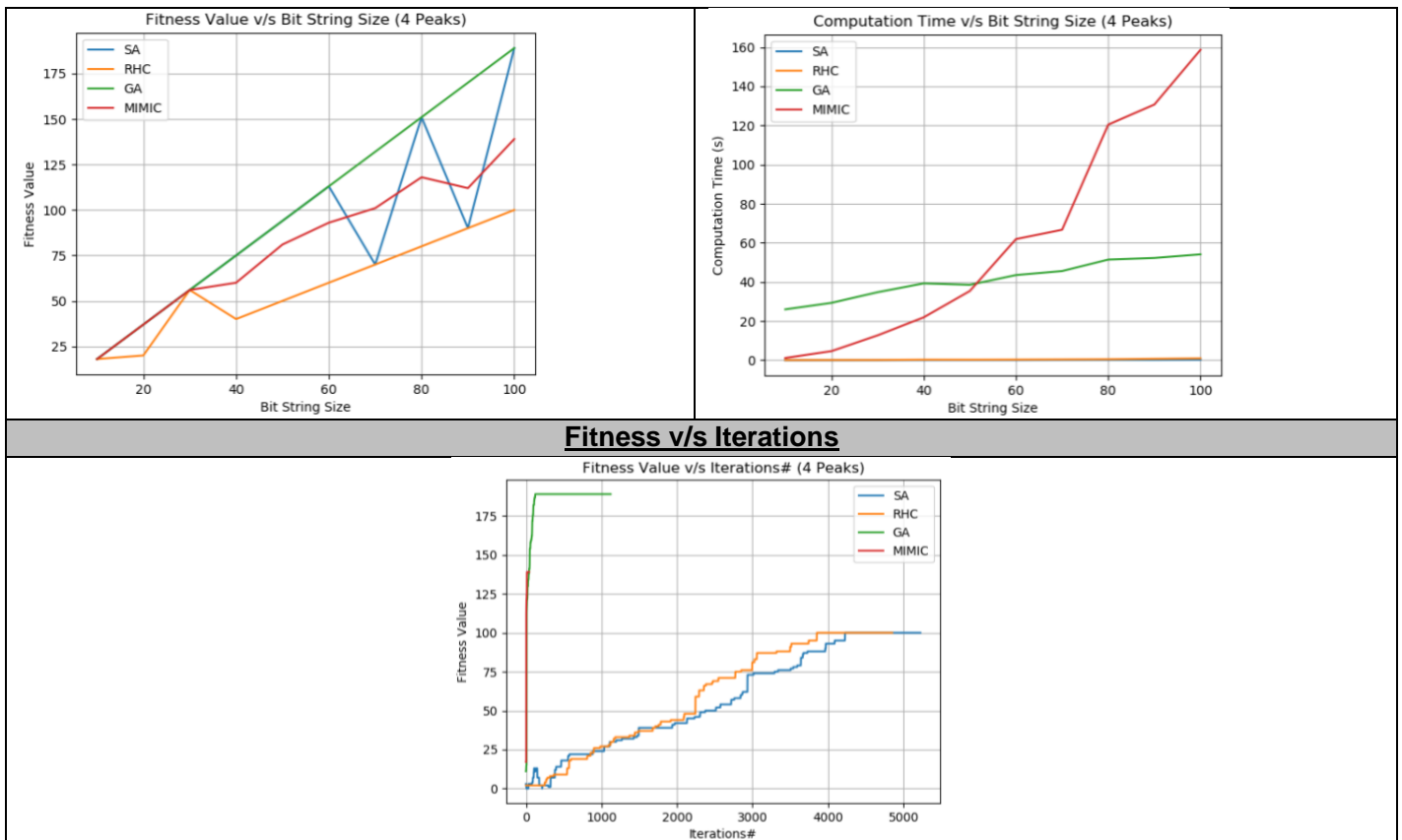
Optimal param value: keep_pct=0.1 and pop_size=500 showed a very good fitness value crossing the local optimal space.



Now with the tuned hyper-parameters we ran each of the Randomized Optimization algorithms to analyze their relative strength and weaknesses.

Fitness v/s Bit String Size

Computation Time (s) v/s Bit String Size



a) GA has outperformed all other algorithms reaching the global optimal fitness value consistently for all input sizes through uniform crossover (default in mlrose). Example of few fitness value reached are...

- String(100): global optima=189
- String(80): global optima= $2 \times 80 - (8+1) = 160-9 = 151$
- String(40): global optima= $2 \times 40 - (4+1) = 80-5 = 75$

Also reached the best fitness value in least number of iterations compared to other algorithms but at the cost of computation time. It took considerably high clock time (after MIMIC) to reach the global optima

b) MIMIC's performance is also consistent with crossing the local optimal fitness value and staying in between the local and global optimal fitness value, but it could not reach the global optima for larger strings. For strings with 10 to 30 length it did reach the global optima.

The 2nd best performance is shown by MIMIC but at the cost of highest run time.

c) SA with high temperature coefficient (init_temp=10) was able to jump over the large basin of attraction of local optima with more randomness and more exploration power. This helped SA to reach global optimal fitness for a variety of input sizes (not all) though it's not a consistent performance throughout. This is a very interesting power of Simulated Annealing.

Computation time for SA is very minimal due to the simplicity of its algorithm same as RHC but with added feature of Exploitation/Exploration.

d) RHC was only able to reach the global maxima for small input size line $N=30$. This is because the basin of attraction is considerably larger (for global maxima) for strings with shorter length so RHC could reach them and move to the global maxima.

Run time is also very small due to its simple algorithm.

Continuous Peaks:

In the Continuous Peaks problem, rather than forcing 0's and 1's to be at opposite ends of the solution string, they are allowed to form anywhere in the string.

For this problem, a reward is given when there are greater than T contiguous bits set to 0, and greater than T contiguous bits set to 1. Here, T is calculated as a percentage of N (bit string length).

Fitness function for Continuous Peaks optimization problem. 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 $(max_run(0, x) > T \text{ and } max_run(1, x) > T)$; and
- $R(x, T) = 0$, otherwise.

The above description of Continuous Peaks has been taken from [2] & [3]

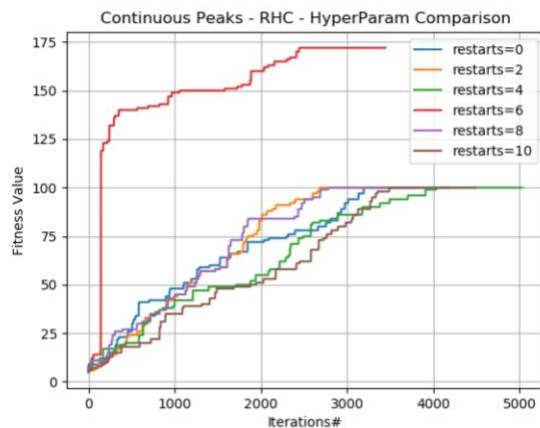
Problem configuration:

Here we shall consider a bit string of length (i.e., N) = 100 and T as 10% of the bit length. Therefore, the MLRose parameter t_pct is set to 0.1 (i.e., 10 percent on N).

The below graphs represent the Fitness Value achieved by four RO algorithms when ran on varying hyper-param values. We shall select those hyper-param values that reached highest Fitness Value at lower Iterations.

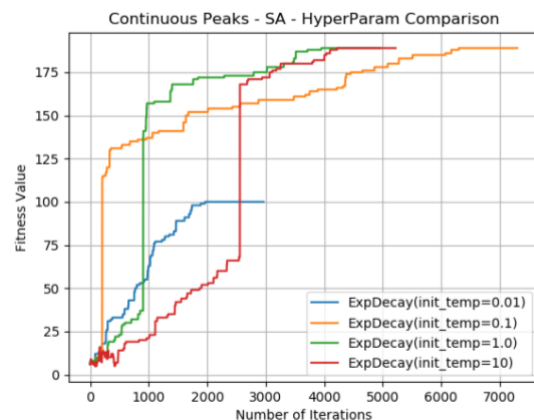
RHC shows a decent fitness value with more restarts (i.e., 6).

Optimal Hyper-param: restarts=6



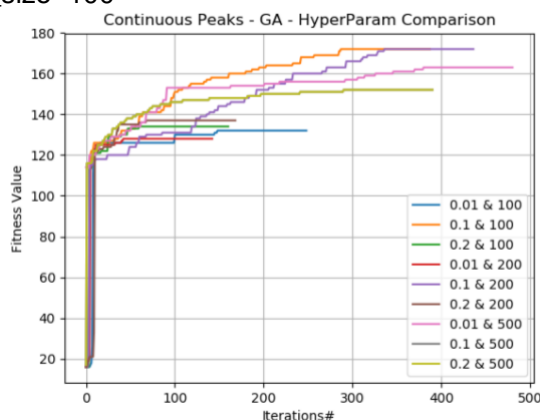
SA performed very well with quiet few parameters but we shall consider the one which reached the max fitness within least iterations.

Optimal Hyper-param: init_temp=1.0



GA looks to be performing best for 0.1/100 (orange curve),

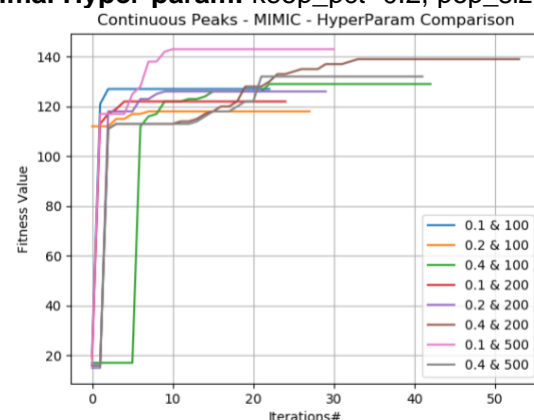
Optimal Hyper-param: mutation_prob=0.01, pop_size=100



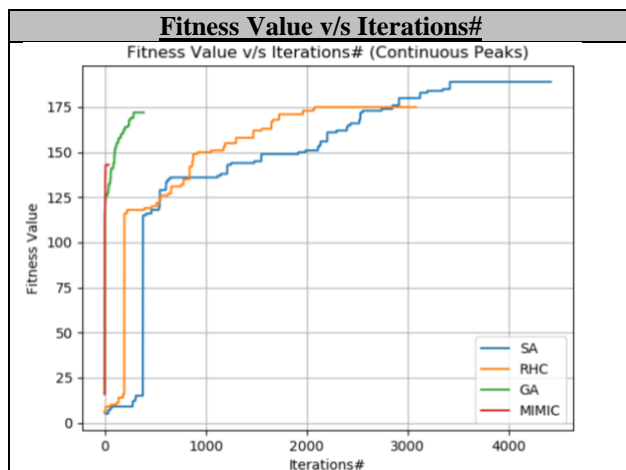
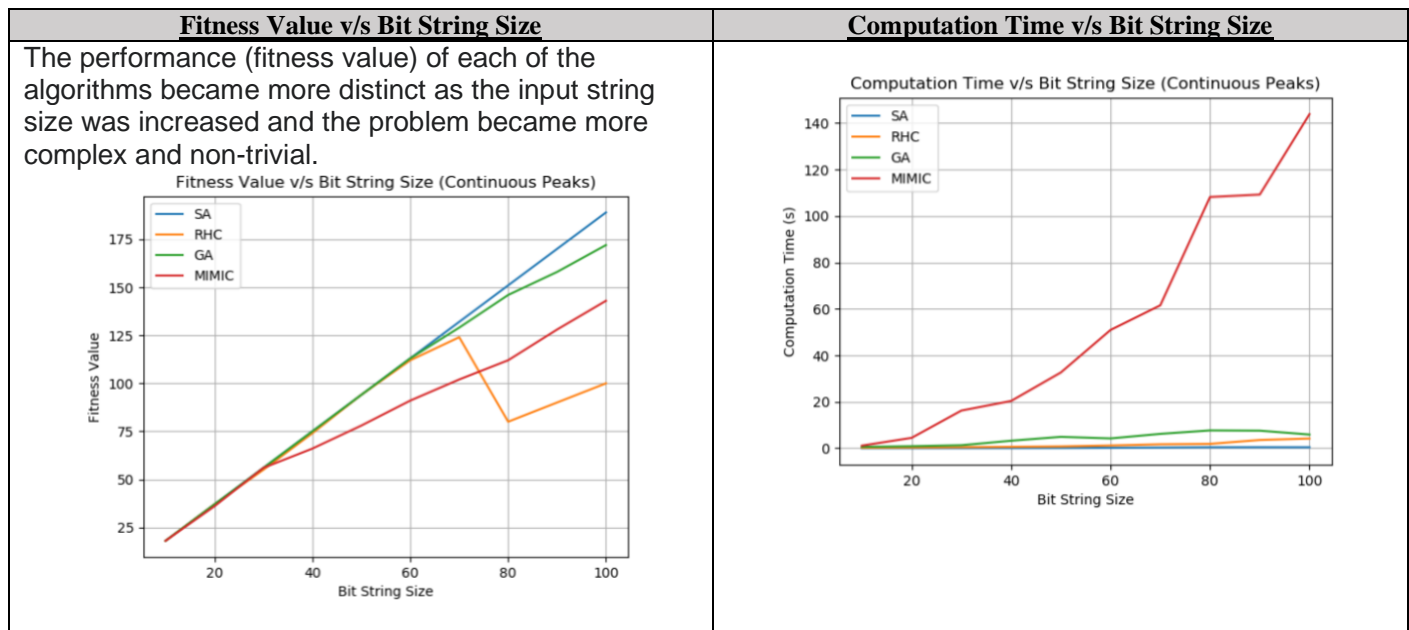
MIMIC's best performance looks to be at 0.2/100.

MIMIC performs poorly for problems which does not have an underlying structure.

Optimal Hyper-param: keep_pct=0.2, pop_size=100



Now with the tuned hyper-parameters we ran each of the Randomized Optimization algorithms to analyze their relative strength and weaknesses.



[Simulated Annealing is the best performer](#) for this problem as it is meant for problem that is more suited for Hill Climbing in addition to the exploration property to avoid getting stuck in local maxima. It also shows a very low computation time. Also, number of iterations is very high as Continuous Peaks requires huge number of iterations to reach its global optimal.

Since this problem is more suited for hill climbing optimization, so Random Hill Climbing is the 2nd best performer after Simulated Annealing. RHC also looks to be little slower than SA probably due to the number of restarts (6 restarts used).

Here the no. of iterations required by Continuous Peaks is pretty high, which is why MIMIC is extremely slow. For MIMIC, best performing situation is less iterations but more time-consuming evaluations.

Knapsack:

Knapsack is a classic NP-Complete problem which has no known polynomial algorithm that can tell, given a solution, whether it is optimal or not.

The problem can be expressed as a set of n items numbered from 1 up to n each with a weight and a value, along with a maximum weight capacity W .

The goal is to maximize $\sum_{i=1}^n v_i x_i$ such that $\sum_{i=1}^n w_i x_i \leq W$.

The above description of Knapsack is taken from [4]

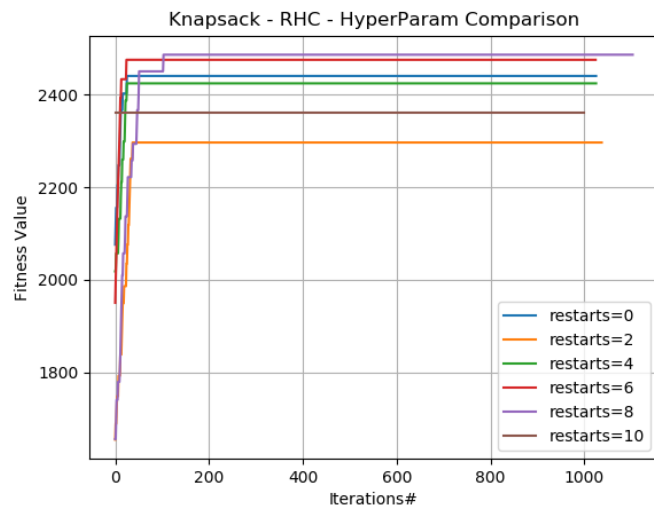
Problem configuration:

To obtain a non-trivial problem size the length of the knapsack is kept at $n=100$. The list of weights () is generated from uniform random integer 20 to 50 and list of value () from uniform random integer 30 to 50. The maximum weight is kept as 60% of total .

The below graphs represent the Fitness Value achieved by four RO algorithms when ran on varying hyper-param values. We shall select those hyper-param values that reached highest Fitness Value at lower iterations.

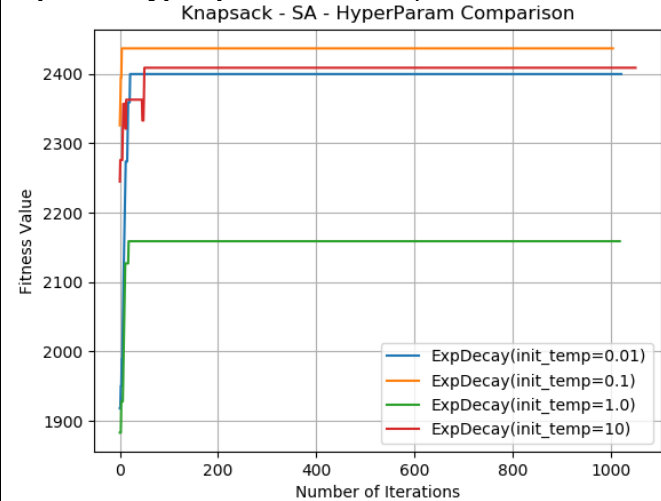
For **Random Hill Climbing** a significant number of restarts were required to reach a descent Fitness Value.

Optimal Hyper-param: restarts=8



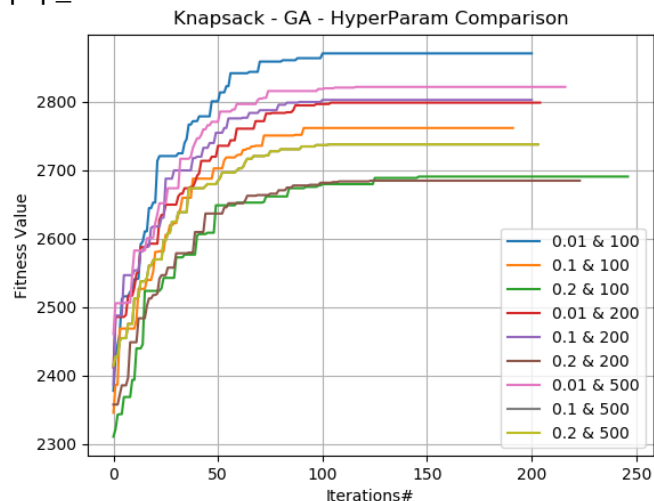
Simulated Annealing performed better at lower initial temperature. May be because Knapsack needs for exploitation of the given list of all weights/values to find the optimal set.

Optimal Hyper-param: init_temp = 0.1



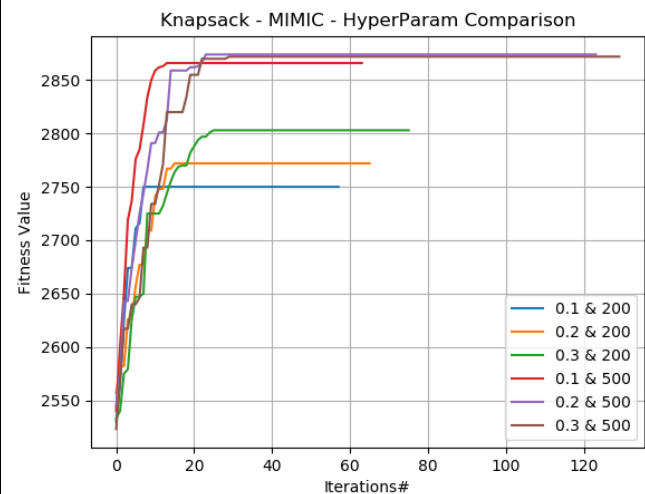
For **Genetic Algorithm** performed its best at a very low mutation probability and small population.

Optimal Hyper-param: mutation_prob=0.01 & pop_size=100

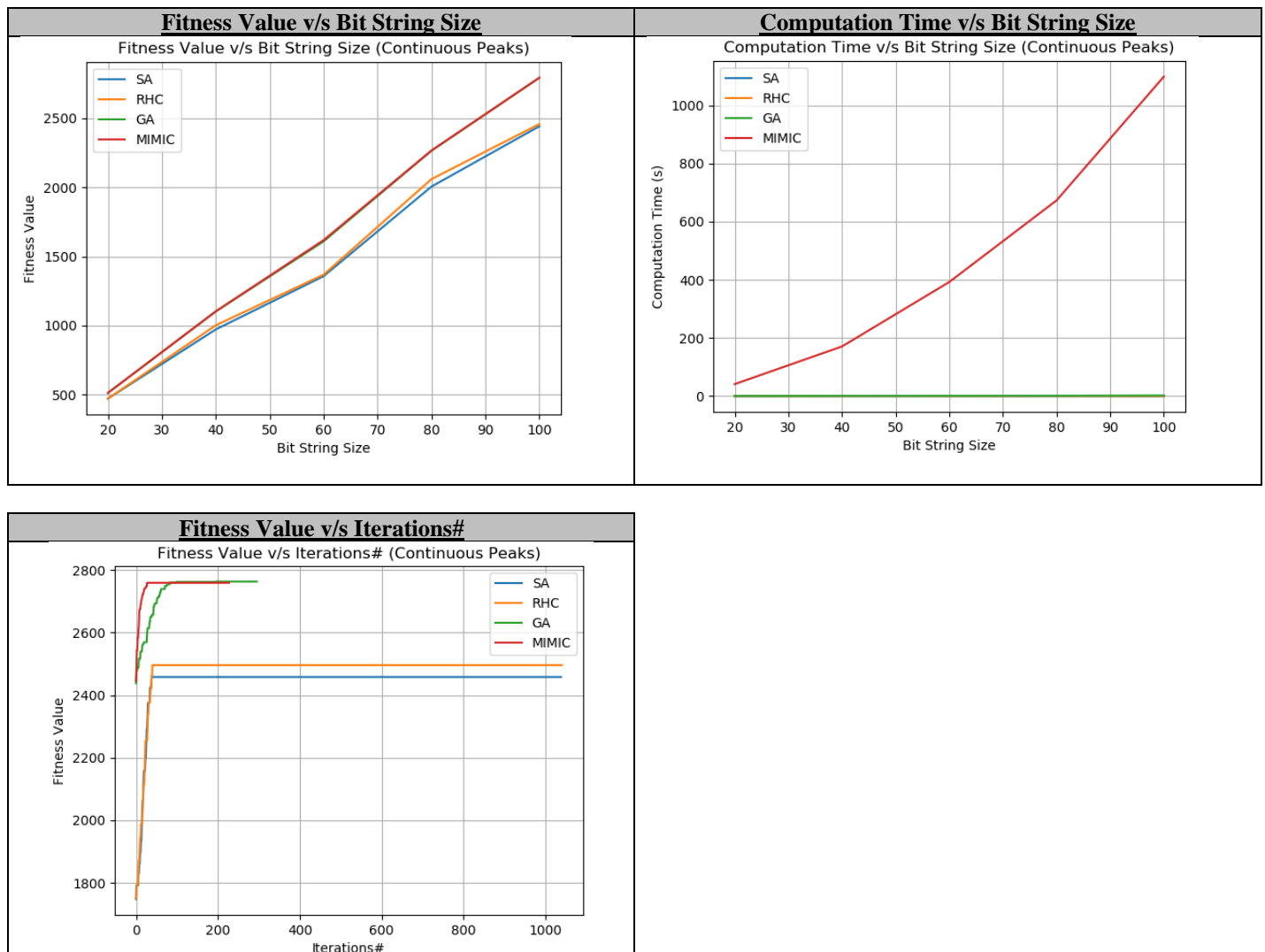


MIMIC looks to be performing best for a variety of parameter values but we shall pick the one with least number of iterations.

Optimal Hyper-param: keep_pct=0.2 & pop_size=500



Now with the tuned hyper-parameters we ran each of the Randomized Optimization algorithms to analyze their relative strength and weaknesses.



This is the problem where [MIMIC really performed as best of the lot](#) (highest fitness value achieved) with lowest number of iterations. This shows that MIMIC can really outperform for optimization problems which needs longer evaluation time but less iterations.

Random Hill Climbing and Simulated Annealing shows the lowest fitness performance as they are meant to solve problems of hill climbing nature which Knapsack is not.

Genetic Algorithm, RHC and SA computation time is very low compared to the exponential time needed by MIMIC. That is why the computation time of GA, RHC and SA appears almost a straight line near zero in the graph in order to visually fit the exponential time of MIMIC.

Part 2: Optimizing Weights in Neural Networks

We shall be using the following three Random Search algorithms to find optimal weights to train our Neural Network used in assignment 1 on the Breast Cancer Wisconsin (Diagnostic) Dataset.

> Random Hill Climbing (Restarts) > Simulated Annealing > Genetic Algorithm

Benchmarking:

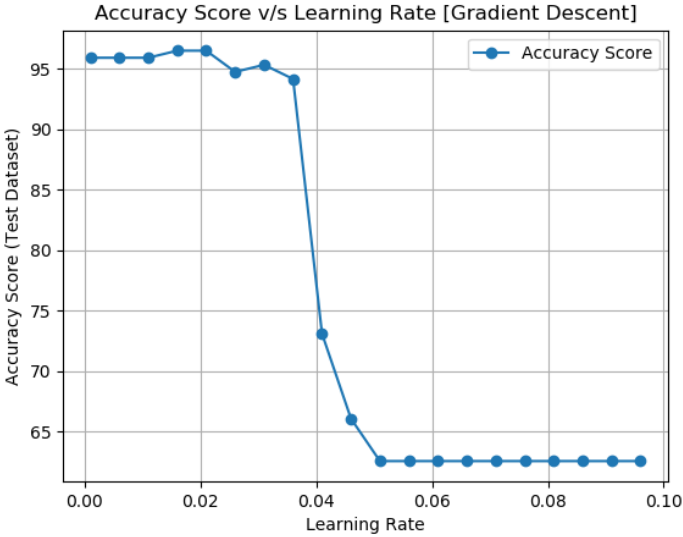
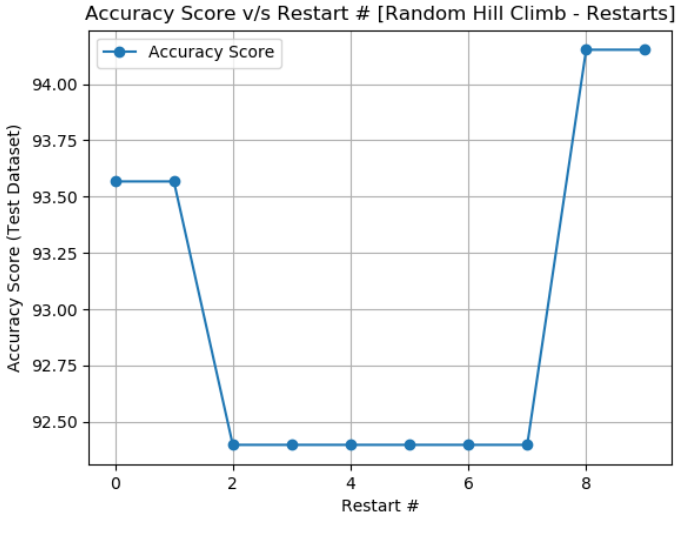
The neural network structure done in assignment 1 shall be taken as a benchmark.

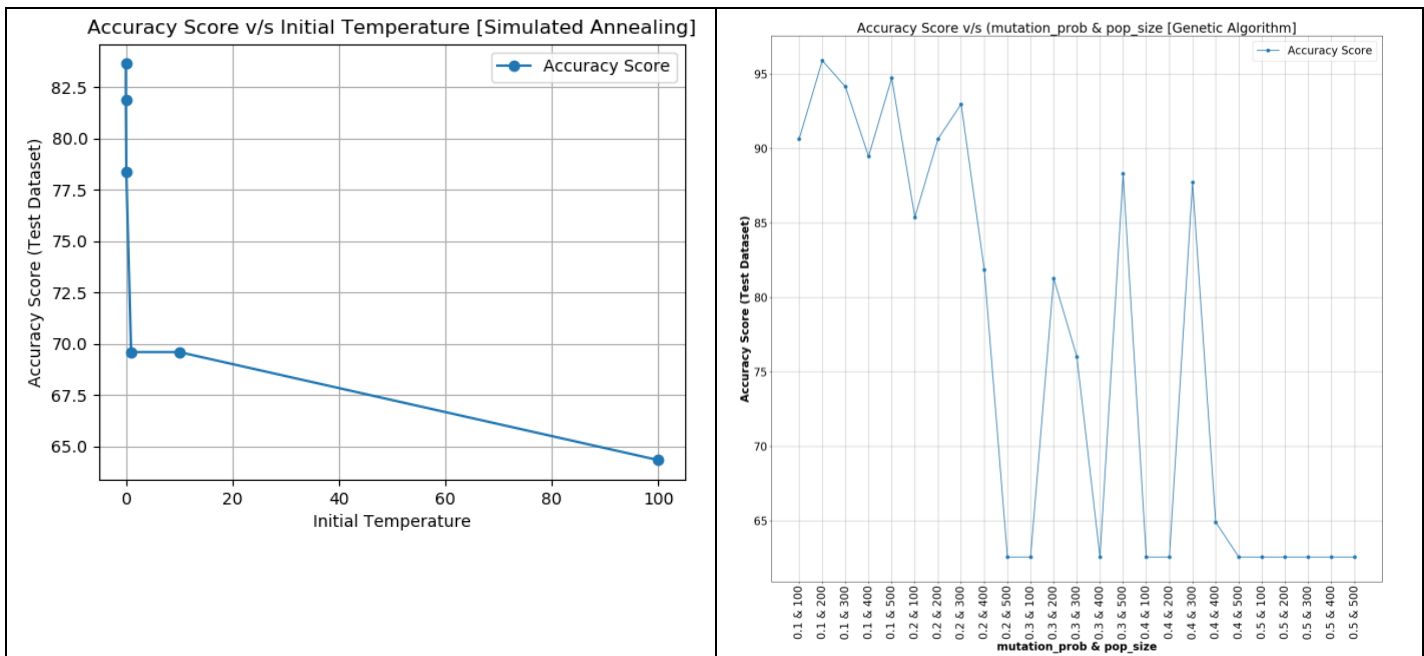
Since Assignment 1 was done using scikit-learn library, there are some differences in output with mlrose. So, I will be doing the benchmarking in mlrose using same structures as in assignment 1:

1. the same network structure (2 hidden layers of 4,3 nodes respectively),
2. same activation function as 'ReLU (i.e., Rectified Linear Unit activation function) and
3. using "Gradient Descent" algorithm.

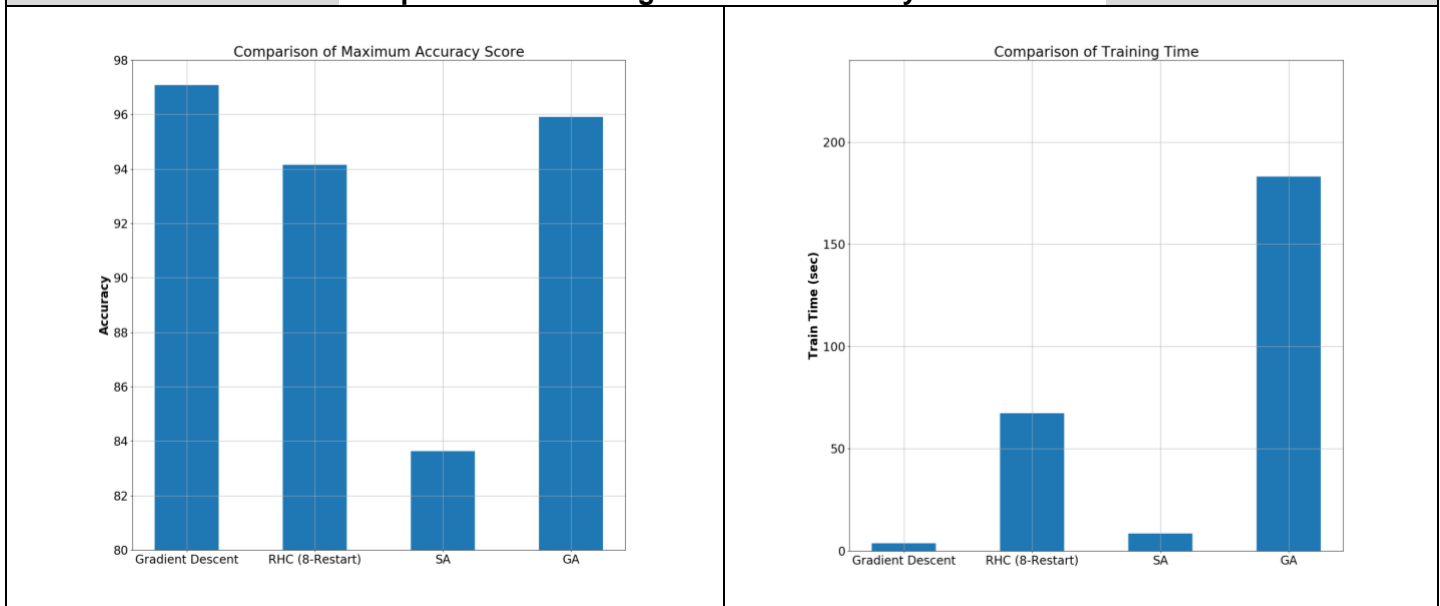
Approach:

1. Run the baseline model (gradient descent) with varying Learning Rate and capture the Learning Rate with highest Test Accuracy Score. This gives us a correctly tuned baseline model.
2. Use the tuned Learning Rate (with the baseline model) as constant for the 3 Random Search based NN models.
So, this way we have built a tuned Neural Network model (the baseline) that we can run with different Random Search Algorithms and compare their relative performances.
3. Now we shall tune the hyper-parameters of each of the RO algorithms by running against a set of possible values and select the best performing parameter based on Test Accuracy Score.
4. We won't be doing any Learning curve or Cross-Validation curve to check "bias-variance trade-off" as we have already done that exercise in Assignment 1.

Hyper-Parameter tuning based on Accuracy Score on Test Dataset																																															
<p>For the Baseline NN model at <u>learning_rate=0.02</u> the accuracy score is maxed with 97.08</p>  <table border="1"><caption>Accuracy Score v/s Learning Rate [Gradient Descent]</caption><thead><tr><th>Learning Rate</th><th>Accuracy Score (Test Dataset)</th></tr></thead><tbody><tr><td>0.00</td><td>95.5</td></tr><tr><td>0.01</td><td>95.5</td></tr><tr><td>0.02</td><td>95.5</td></tr><tr><td>0.03</td><td>95.5</td></tr><tr><td>0.04</td><td>95.5</td></tr><tr><td>0.05</td><td>95.5</td></tr><tr><td>0.06</td><td>95.5</td></tr><tr><td>0.07</td><td>95.5</td></tr><tr><td>0.08</td><td>95.5</td></tr><tr><td>0.09</td><td>95.5</td></tr><tr><td>0.10</td><td>95.5</td></tr></tbody></table>	Learning Rate	Accuracy Score (Test Dataset)	0.00	95.5	0.01	95.5	0.02	95.5	0.03	95.5	0.04	95.5	0.05	95.5	0.06	95.5	0.07	95.5	0.08	95.5	0.09	95.5	0.10	95.5	<p>RHC's accuracy was highest at 94.15 with <u>restarts=8</u>. INPUT learning_rate=0.02</p>  <table border="1"><caption>Accuracy Score v/s Restart # [Random Hill Climb - Restarts]</caption><thead><tr><th>Restart #</th><th>Accuracy Score (Test Dataset)</th></tr></thead><tbody><tr><td>0</td><td>93.6</td></tr><tr><td>1</td><td>93.6</td></tr><tr><td>2</td><td>92.4</td></tr><tr><td>3</td><td>92.4</td></tr><tr><td>4</td><td>92.4</td></tr><tr><td>5</td><td>92.4</td></tr><tr><td>6</td><td>92.4</td></tr><tr><td>7</td><td>92.4</td></tr><tr><td>8</td><td>94.15</td></tr><tr><td>9</td><td>94.15</td></tr></tbody></table>	Restart #	Accuracy Score (Test Dataset)	0	93.6	1	93.6	2	92.4	3	92.4	4	92.4	5	92.4	6	92.4	7	92.4	8	94.15	9	94.15
Learning Rate	Accuracy Score (Test Dataset)																																														
0.00	95.5																																														
0.01	95.5																																														
0.02	95.5																																														
0.03	95.5																																														
0.04	95.5																																														
0.05	95.5																																														
0.06	95.5																																														
0.07	95.5																																														
0.08	95.5																																														
0.09	95.5																																														
0.10	95.5																																														
Restart #	Accuracy Score (Test Dataset)																																														
0	93.6																																														
1	93.6																																														
2	92.4																																														
3	92.4																																														
4	92.4																																														
5	92.4																																														
6	92.4																																														
7	92.4																																														
8	94.15																																														
9	94.15																																														
<p>Simulated Annealing reached it max accuracy of 83.63 at <u>init_temp=0.001</u>. INPUT learning_rate=0.02</p>	<p>Genetic Algorithm reached max accuracy of 95.91 at <u>mutation_prob=0.1</u> and <u>pop_size=200</u>. INPUT learning_rate=0.02</p>																																														



Comparison of Training Time and Accuracy on Test Data:



We can see from the above graph that Gradient Descent shows the best Accuracy Score on the test data out of all the four algorithms with the least training time. Though Gradient descent can fall for Local minima, but versions like stochastic descent can add more randomness which can help to locate the steep descent to a global optimum.

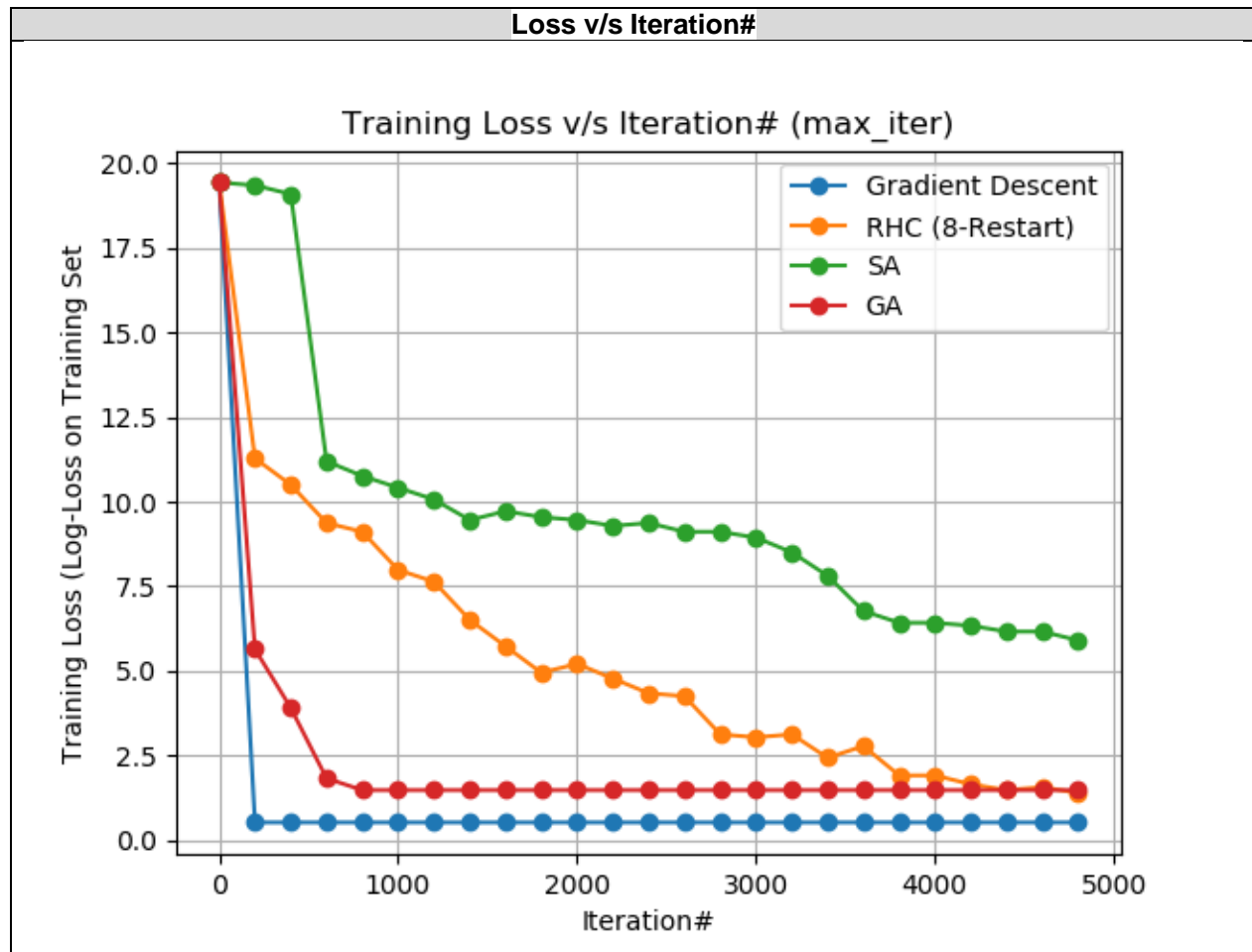
The reason for using an “ReLU” activation function as it helps to achieve faster learning of the models by handling the Vanishing Gradient problem. Vanishing Gradient is problem that happens in Back Propagation methods which makes it very difficult the learning process very difficult for the earlier networks. That’s why ReLU shows faster training time than Sigmoid activation function.[5]

Among the three Randomized algorithms, Genetic Algorithm comes best with an Accuracy Score of 96% (approx.) but at the expense of very high Training Time. The high Training Time may be due to the no. of function evaluations required is proportional to the population size. The reason of high accuracy score may be due to better exploration power of the solution space (to find global optima) by GA compared to SA or RHC.

Loss v/s Iteration plot:

Using the `log_loss` function provided by scikit-learn, I tried to capture the loss curve by varying the iteration over multiple runs to see the progression of each of the algorithms minimize the loss function.

We can compare the Loss minimization to the accuracy score for each of the RO algorithms and see that algorithms with high Accuracy Score have better loss minimization than the rest.



Citations:

- [1] *Randomized Local Search as Successive Estimation of Probability Densities* by Charles L. Isbell, Jr.
- [2] *Removing the Genetics from the Standard Genetic Algorithm* by Shumeet Baluja & Rich Caruana
- [3] <https://mlrose.readthedocs.io/en/stable/source/fitness.html>
- [4] https://en.wikipedia.org/wiki/Knapsack_problem#0-1_knapsack_problem
- [5] <https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/#:~:text=In%20a%20neural%20network%2C%20the,or%20output%20for%20that%20input.>