Randomized Optimization

CS 7641 – Machine Learning

Manish Mehta

Introduction

The purpose of this project is to explore and apply some of the randomized optimization techniques for two different problem statements – one is to determine the weights in a neural network that was used on the Phishing dataset in the first assignment on Supervised Learning, and other is to create three optimization problem domains and apply each of the four search techniques to these problems. Same as before, for the purpose of this assignment, we have selected the data on "Phishing Websites", as it is of the most attractive areas of interest and has a large-scale impact for any corporation and organization[1,2,3,4]. The aim is to understand, implement and analyze each of these search techniques instead of backpropagation to find the weights in the neural network on Phishing dataset. Also, we apply each of them on the three optimization problems, eventually comparing & contrasting all of the algorithms against each other in terms of performance, accuracy, speed and efficiency. We have mostly used F1 score as a metric for estimating the accuracy/performance of the neural network model across these techniques and how quickly each of the randomized optimization algorithms reaches the optimal value for the fitness-function scenario. Along with that, we also graphically report the training, testing accuracies, and fit times against number of iterations for each of the techniques for the sake of completion.

Analysis and Randomized Optimization Algorithms

The Randomized Optimization techniques we looked at included Randomized Hill Climbing (RHC), Simulated Annealing (SA), Genetic Algorithm (GA), and MIMIC. The first part of the assignment applied the first three algorithms to determine the weights in neural network on Phishing Dataset (preprocessed) as obtained from the previous assignment. We then compared current results with the earlier results of backpropagation and also with each other. Care was taken while implementing these algorithms since weights in a neural network are continuous and real-valued instead of discrete. In part two, we will create three optimization problems and apply each of the 4 algorithms to them, comparing and contrasting their performance on each problem, and which performs better and why. Graphical results are reported to assist in visually understanding each scenario. All analysis was done using Java ABAGAIL Library[5].

Part 1: Neural Networks with Randomized Optimization

In this section we go through the results of applying each of the 3 Randomized learning algorithms to the optimal neural network structure we got from the earlier assignment, which was (100,1), i.e. input layer with 100 neurons and output layer with 1 neuron, with a learning rate of 0.015. We looked for any interesting insights and possible scope of improvement, that could have been implanted given more time and resources. In general, for each case, 80% of the data was used for training and 20% was used for testing the algorithm accuracy. F1 score was used as a metric for understanding the performance of the algorithms.

A feed-forward neural network was used. We have to remember that all the configurations are the same as the earlier assignment, just that now backpropagation was not used for updating the weights. The number of iterations considered were 10, 100, 500, 1000, 2500, 5000, which gives us a nice spread to understand how each algorithm performs. This helped plot a series of curves for each of the algorithms as shown below.

Randomized Hill Climbing (RHC)

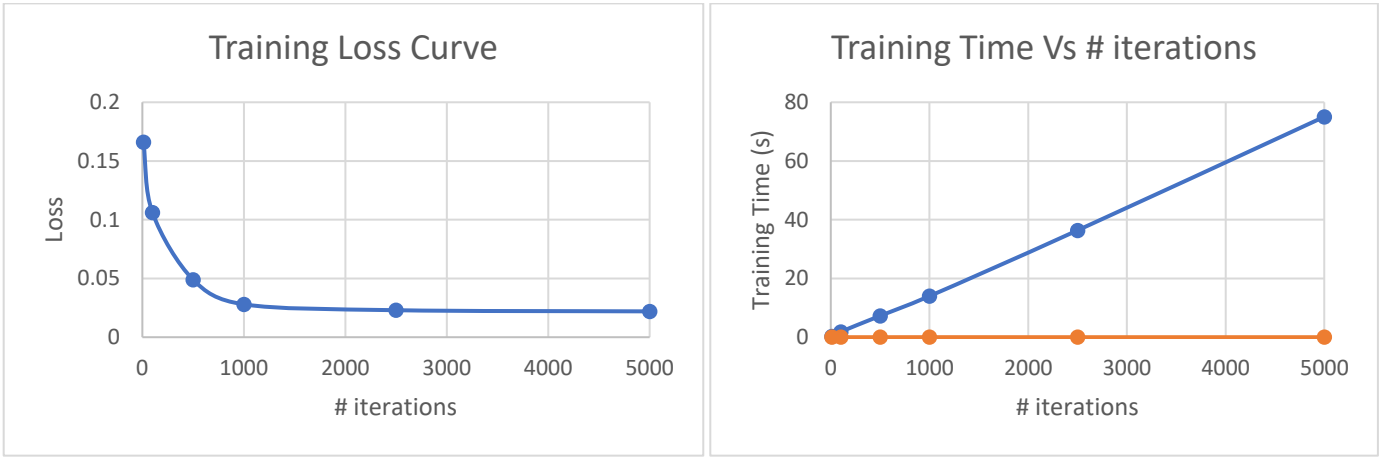


Fig.1 Training Loss and Training Time curves for RHC

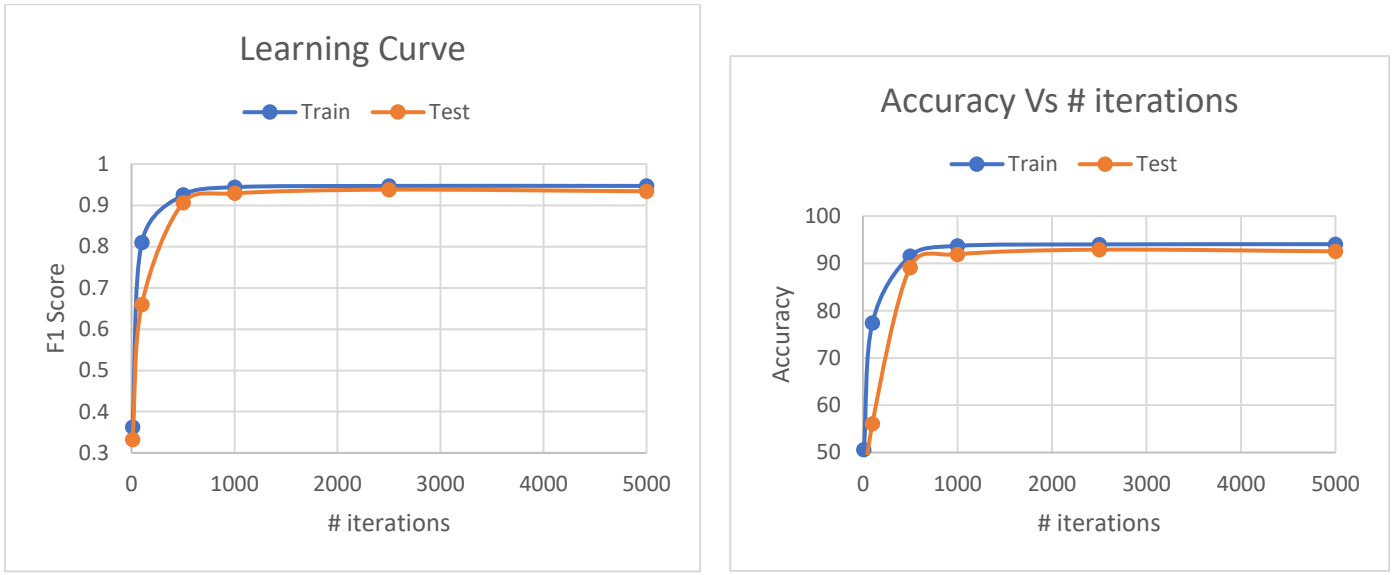


Fig.2 F1 Score and Accuracy Vs Number of Iterations for Training and Testing datasets for RHC

We can observe from above that as the number of iterations increases, the model accuracy (both F1 score and Percentage examples correctly classified) saturates for both training and test sets. Since this was the initial of the analyses, we have shown both the F1 score and the Accuracy (percentage examples correctly classified) for both the training and test data. But in further report, we have only reported F1 score, since the accuracy follows accordingly, and the metric we have been using throughout is F1 score. Also, with an increase in iterations, the loss curve converges slowly towards zero, but with that the training time increases linearly. This behavior in training time and testing time with iteration was similar across all other algorithms too, so for the sake of being concise, it is not shown anywhere else. Given more time and resources, we could have plotted the Loss curve for RHC for different randomized restarts which would indicate that the losses are different for different runs because of the stochastic nature of the algorithm and the tendency to get stuck in the local optima. Observing the Learning curves, we can say that the model is more robust than the backpropagation case, since the training and testing curves converge well, and there are on signs of overfitting yet. Also, observe that the training times are extremely larger than the testing times, which is the innate characteristic of the Neural Networks, and also that the testing time remains unaffected with the number of iterations. A tabulated summary across all the randomized algorithms with the backpropagation will be provided at the end of this section.

Simulated Annealing (SA)

The different values of cooling exponents chosen were {0.05,0.2,0.35,0.5,0.65,0.8,0.95} along with the values of temperatures ranging from {1e1,1e2,1e3,1e4,1e5,1e6,1e7}. The results for each combination for the training and testing scenarios are shown below:

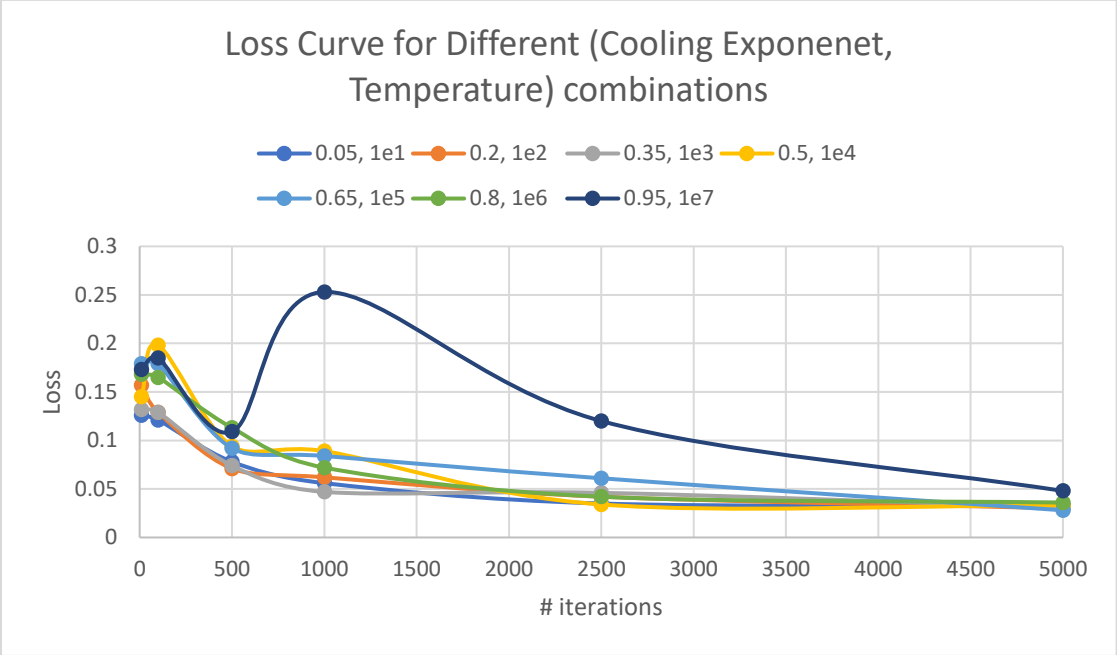


Fig.3 Training Loss curves for SA for various Cooling Exponent, Temperature combinations

In the above plot, we can see that all the plots converge towards zero, with the lowest temperature and cooling exponent combination converging smoothly and slowly, while the highest cooling exponent and temperature combination showing rather erratic jumps and behavior before converging, which is expected of the SA algorithm. (SA can accept locally bad moves with a high probability when the temperature is high. We can observe this behavior in the plot where there are large number of fluctuations initially for high temperature. As the number of iterations increases, the temperature goes down and it becomes increasingly unlikely to accept locally bad moves. This can be observed in the plot for later iterations where loss smoothly decreases). Thus, this is in line with our intuition. Below are the plots for Learning Curve for the Testing and Training datasets (they are plotted on separate plots to represent visually the effect different values of cooling exponents and temperatures on the F1 Score)

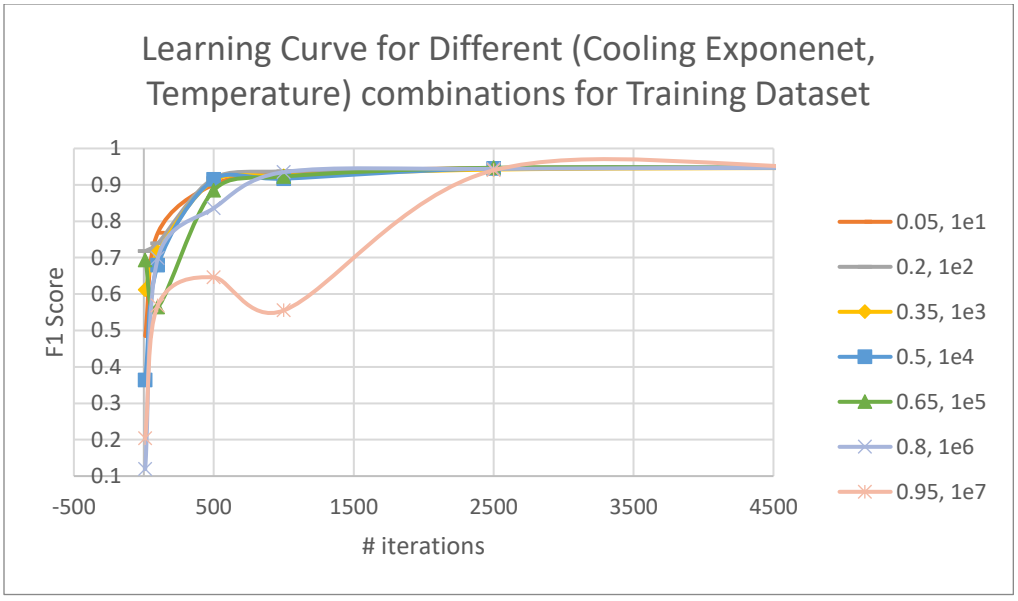


Fig.4 Learning curves for SA for various Cooling Exponent, Temperature combinations for Train Dataset

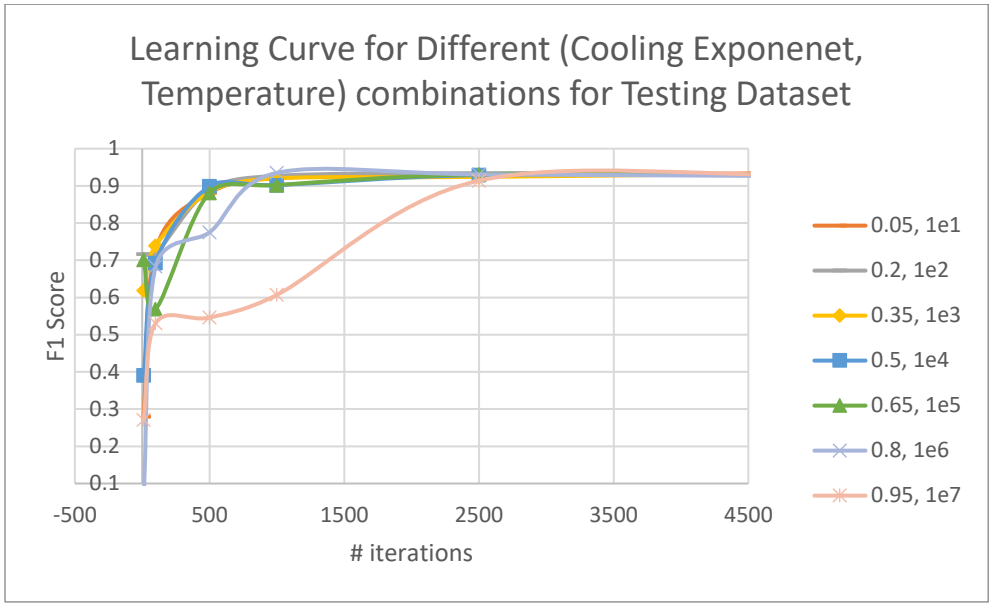


Fig.5 Learning curves for SA for various Cooling Exponent, Temperature combinations for Test Dataset

We observed here that again with an increase in temperature, and the cooling exponent, there was an erratic behavior of the algorithm in converging to the maximum value of F1 score (similar explanation as given in Loss curve). This is intuitive of our understanding of SA algorithm and hence it makes sense to use the optimal combination for determining the weights of the neural network for the Phishing dataset. Overall, the model is robust as there are no signs of overfitting observed. Also, the training times are around the same for each algorithm, with it being slightly less for higher temperature combination since there is lot of jumping around and it converges quickly but does worse in smaller number of iterations, as compared to the smaller values of temperature and cooling exponent (to maintain optimality, the cooling exponent has to be low). Testing time is low and in line with our observations with other algorithms before.

Genetic Algorithm (GA)

The plots for different combinations of population, crossover and mutation used were {10,25,50,75,100,125}, {5,10,30,40,50,60} and {2,5,10,15,20,25} respectively are shown below:

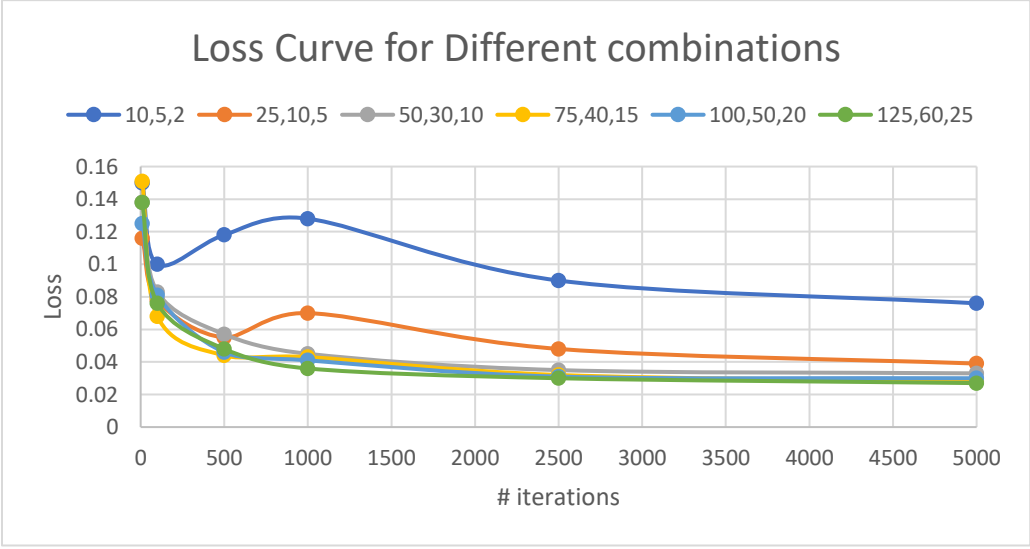


Fig.6 Training Loss curves for GA for various combinations

We can see that larger population sizes have better Loss curves, and they are more smoothly converging towards zero as compared to the smaller population sizes. Also, observe the Learning Curves for the training and testing dataset:

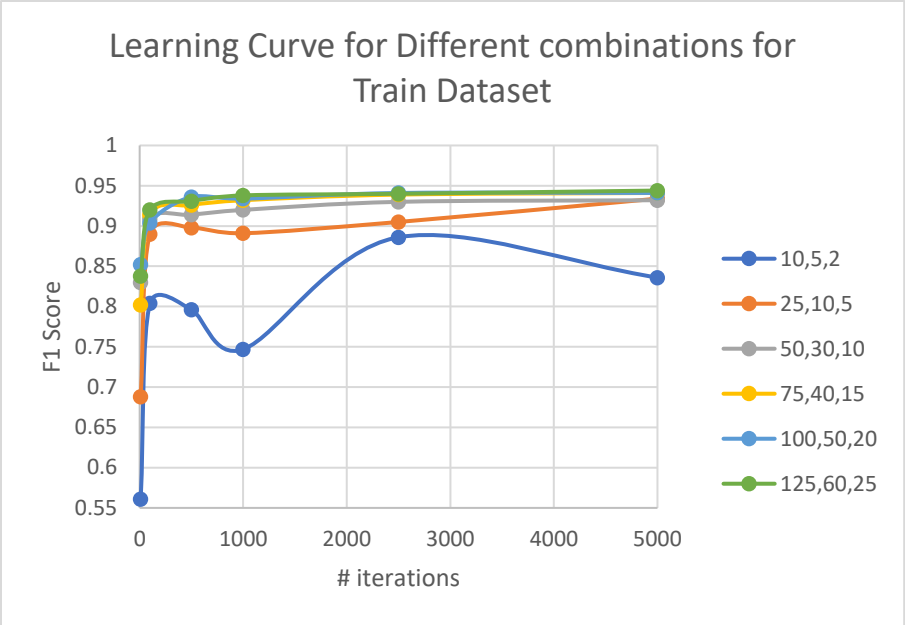


Fig.7 Learning curves for GA for various combinations for Train Dataset

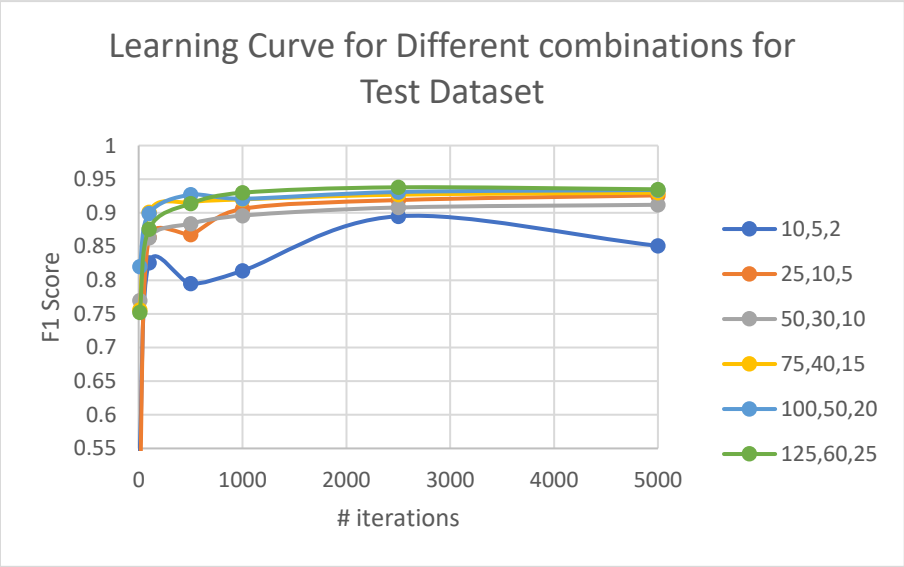
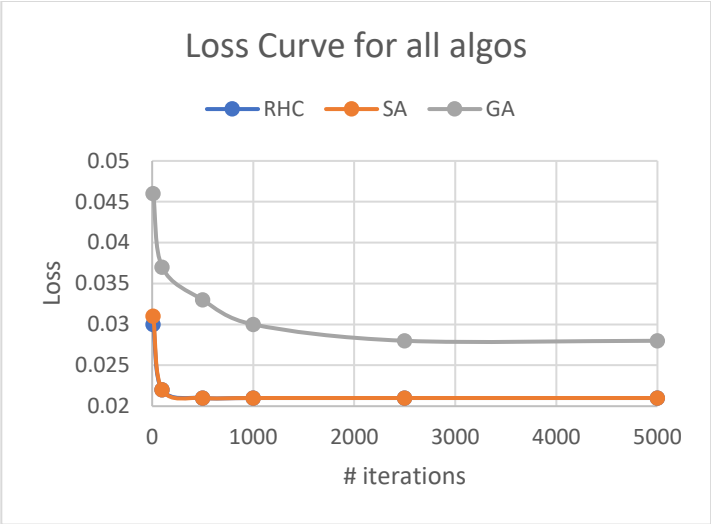


Fig.8 Learning curves for GA for various combinations for Test Dataset

Like SA, GA can also accept locally bad moves, resulting in the fluctuations in losses in the initial iterations. This happens for the smaller population size (that is smaller combination as can be observed from the plots). A plot of the training time (not shown here for the sake of avoiding repetition) gave us the information that the training times for GA went extremely up for larger population sizes. So we have to think of a tradeoff between the training time and the smoothness of the algorithm in achieving the higher accuracy (consider some middle combination).

We observed that for each of the case, convergence is achieved for higher number of iterations. Below are the Loss curve and Learning curve plots for considering optimal parameters for GA and SA and observing how that behaves with number of iterations as compared to RHC



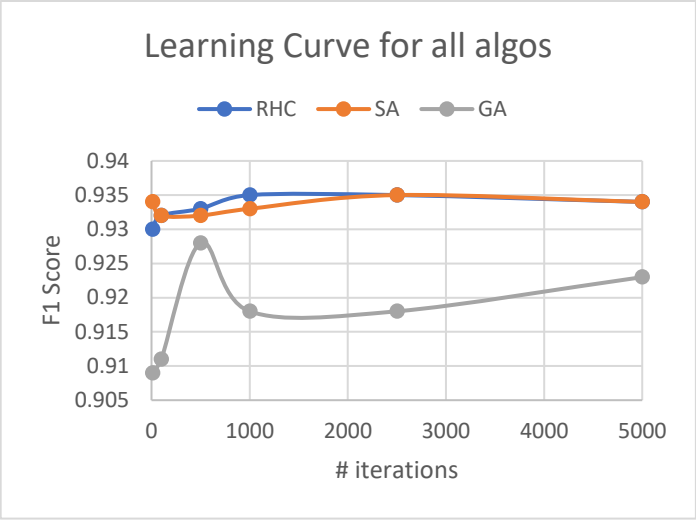


Fig.9 Training Loss and Learning curves for all randomized algorithms

Summary table, is also given below:

Metric	Backpropagation	RHC	SA	GA
Test F1 Score	0.98	0.934	0.934	0.923
Training Time (s)	6.075	77.07	77.255	367.153

Thus, we can see that RHC performs the best in terms of attaining the highest accuracy among the 3 in least amount of time. GA performs the worst, because it is computationally expensive to run it at large population sizes. A single run of RHC requires the least amount of time among the random search algorithms. Both RHC and SA are prone to getting stuck at the local minima, but with optimal parameter values, that can be prevented. Gradient descent, on the other hand, gives the best test accuracy among all the algorithms while also taking the least amount of time to run. This might be because following the direction of steepest descent quickly led to the global optimum. The random search algorithms do not use a gradient and might have explored suboptimal solution spaces. Increasing the dimensionality of the search space might put random search algorithms to an advantage as they have a greater chance of escaping global minima than gradient descent. However, this is at the expense of more computation. To improve the performance of the random search algorithms, cross-validation could be used to tune their hyperparameters (e.g. step size for RHC, initial temperature and cooling exponent for SA, population size, number of crossovers and mutations for GA).

Part 2: Optimization Problems

Three optimization problems have been presented in this section, each of which highlight the importance of using each of the SA, GA and MIMIC over the others. The problems selected are Travelling Salesman Problem, Continuous Peaks problem and Four Peaks problem.

Travelling Salesman Problem (Highlights GA)

Travelling Salesman problem is a classis NP hard problem. The objective is to find the shortest round trip between different points, while visiting each point only once. It is a problem with practical applications in transportation and travel planning. Here, we take different combinations of N = {10,20,30,40,50} and number of iterations for observing the performance of various algorithms:

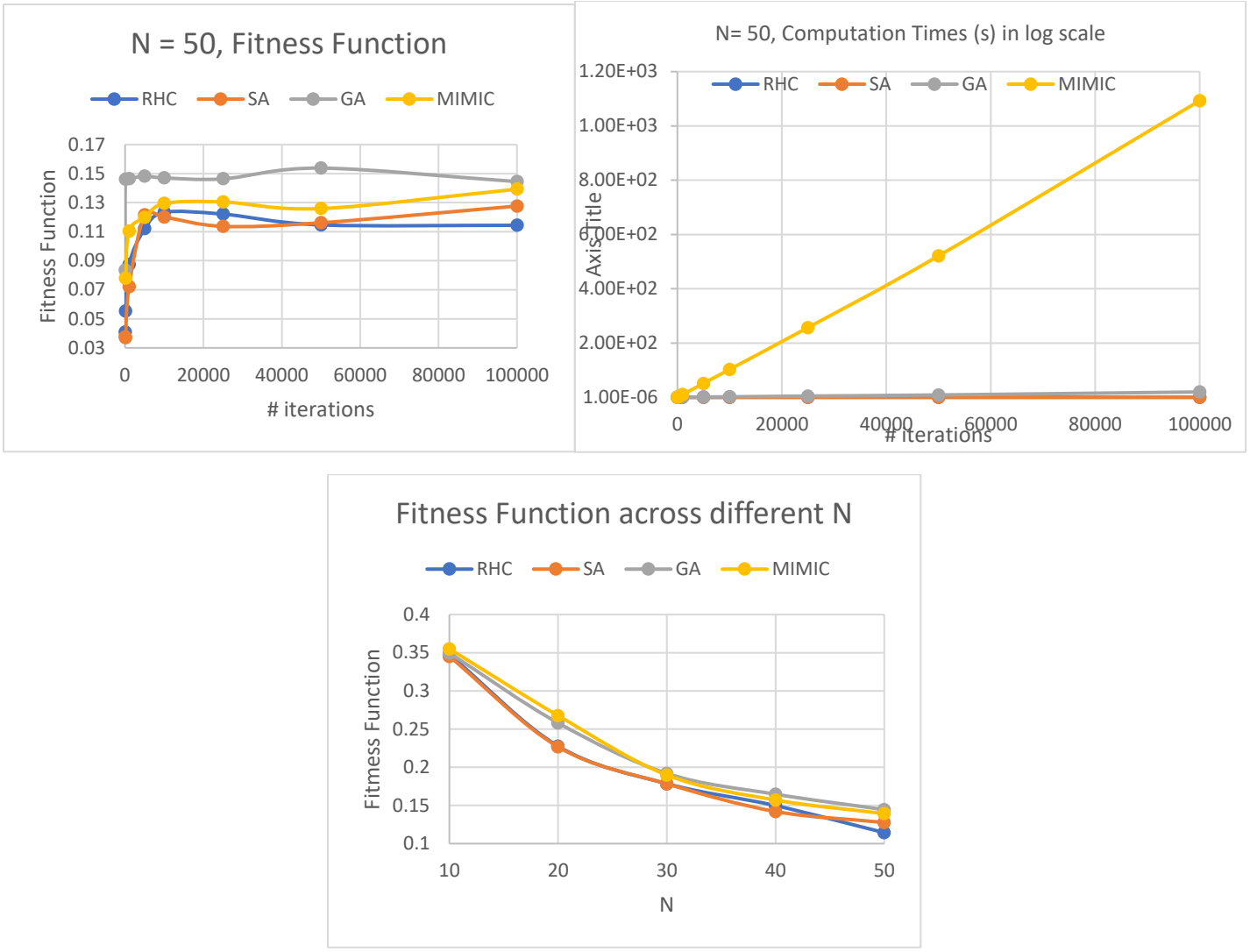
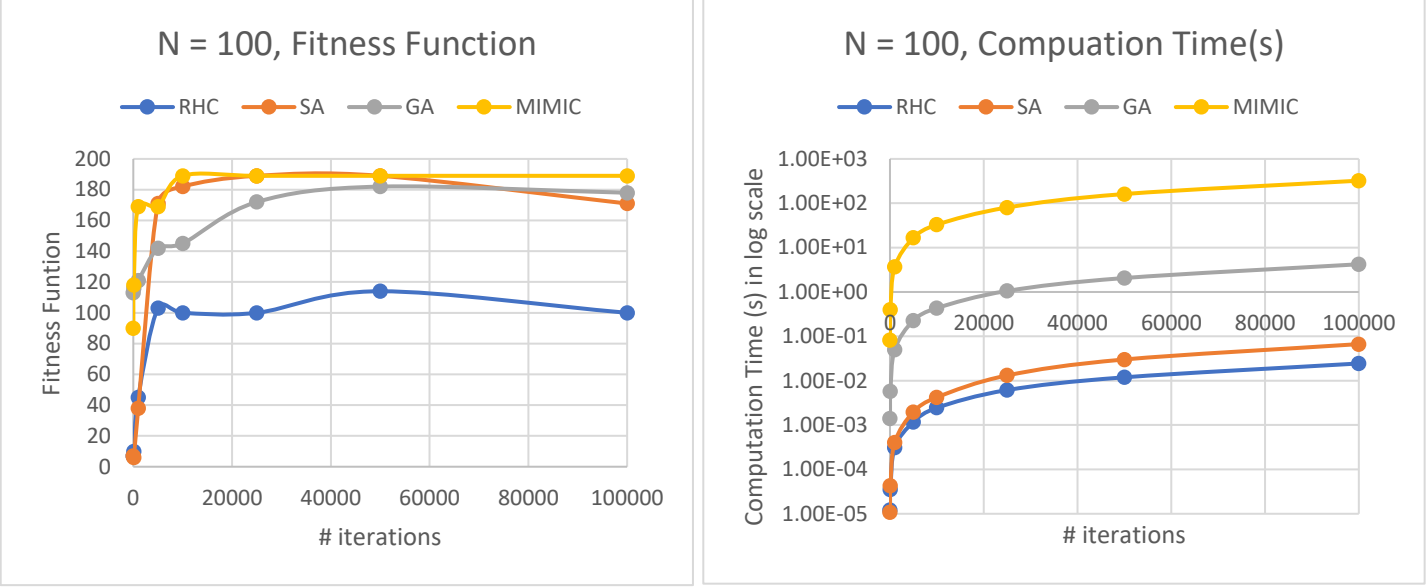


Fig.10 Fitness Function and Computation Times(s) for all algorithms, across different iterations and different N values

Thus, we can see that Genetic Algorithm performs the best among all. The plots show that it performs better for a fixed N across all iterations, and for a fixed iteration(=100,000) across different N values. Computation time surely is more than RHC and SA, but significantly less than MIMIC.

Continuous Peaks Problem (Highlights SA)

Continuous peaks problem is the optimization problem that tries to find out highest peak (global optima) with respect to other peaks (local optima). It is an extension of Four Peaks (explained in next section). Rather than forcing 1's and 0's to be at the ends of the string, they are allowed to form anywhere in the string. A reward is given when there are greater than T contiguous bits set to 0, and greater than T contiguous bits set to 1. It has many local maxima. In ABAGAIL, we take N={20,40,60,80,100} and T=10, results are shown below:



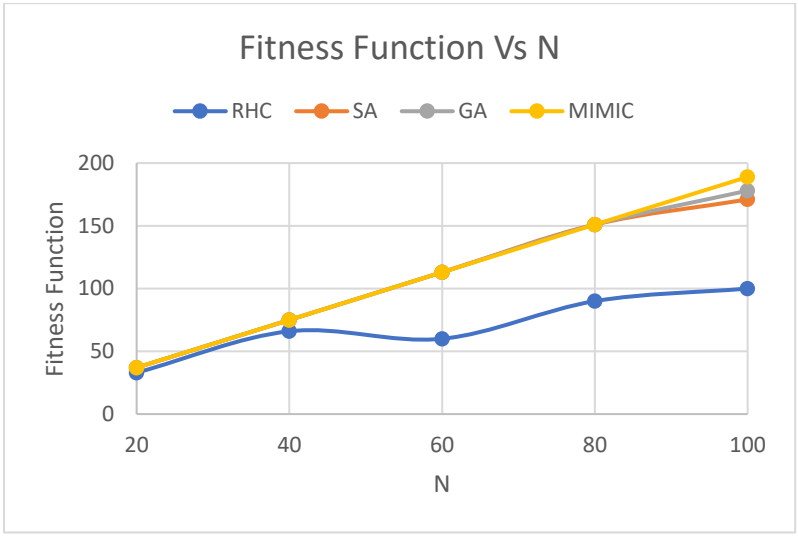


Fig.11 Fitness Function and Computation Times(s) for all algorithms, across different iterations and different N values

Clearly, SA is the winner. Here, we observe the results for a fixed N(=100) and varying number of iterations and for fixed number of iterations(=100,000) and varying N. In either case, SA performs equally well as MIMIC, but at highly reduced computation times. At higher values of N, it's possible that SA did not explore the solution space enough at higher temperatures and converged to the local optimum as the temperature decreased. But overall, SA is the winner.

Four Peaks Problem (Highlights MIMIC)

The problem is defined as: Given an N-dimensional bitstring x, the four peaks evaluation function is given by:

$$f(\mathbf{x}, T) = \max\{tail(0, \mathbf{x}), head(1, \mathbf{x})\} + R(\mathbf{x}, T)$$

where

$$tail(0, \mathbf{x}) = \text{number of trailing 0's in } \mathbf{x}$$

$$head(1, \mathbf{x}) = \text{number of leading 1's in } \mathbf{x}$$

$$R(\mathbf{x}, T) = \begin{cases} N, & \text{if } tail(0, \mathbf{x}) > T \text{ and } head(1, \mathbf{x}) > T \\ 0 & \text{otherwise} \end{cases}$$

There are two global maxima for this function. They are achieved either 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 that occur

with a string of all 1's or all 0's. For large values of T, this problem becomes increasingly more difficult because the basin of attraction for the inferior local maxima become larger. The plots for varying N={20,40,60,80,100,120} and T=5 for various iterations is shown:

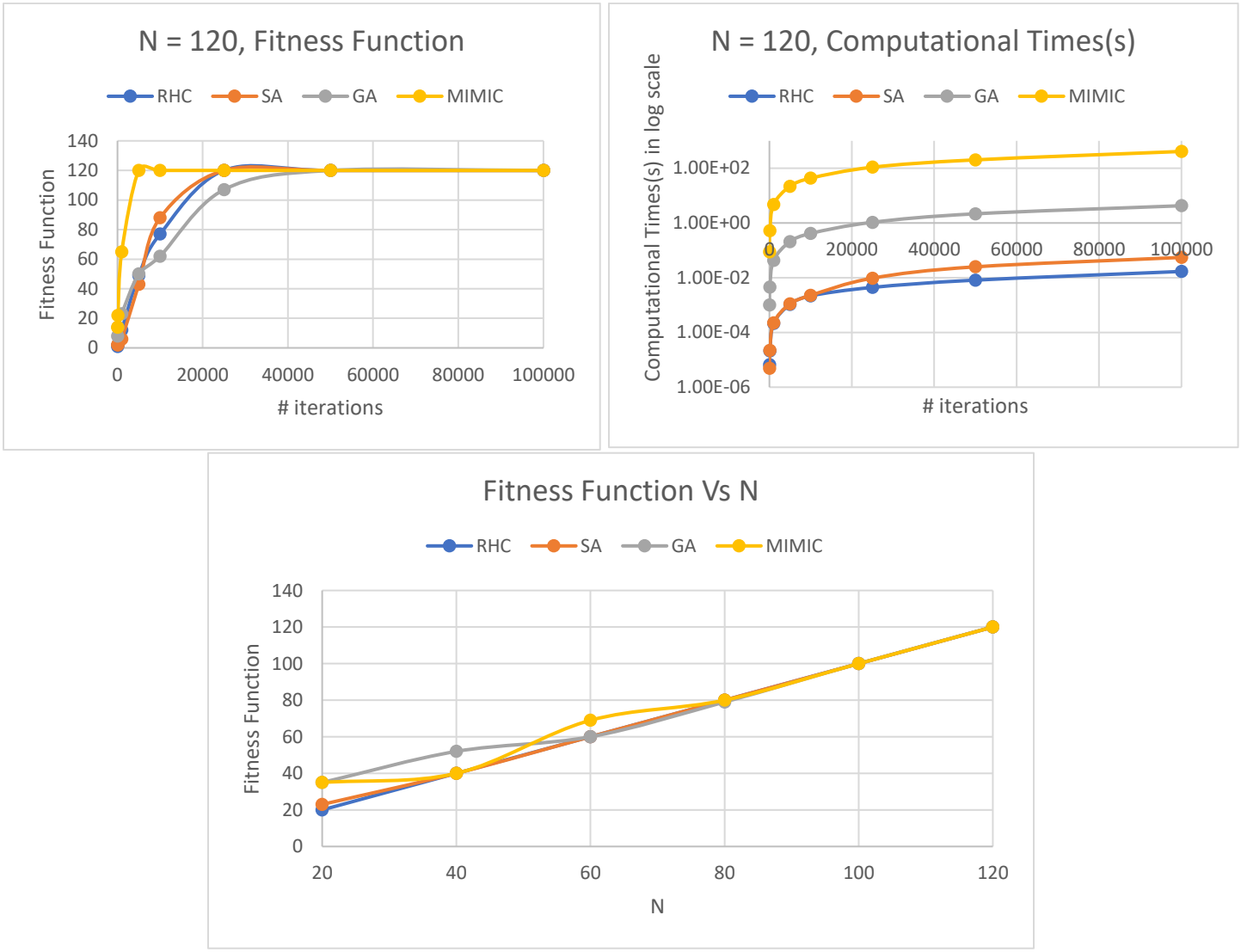


Fig.12 Fitness Function and Computation Times(s) for all algorithms, across different iterations and different N values

From above, we can see that MIMIC is the winner. We varied the number of iterations keeping N constant (=120) and then varied N keeping number of iterations at 100,000. We see that in either case, the fitness function attains maximum super quickly with MIMIC. The MIMIC algorithm consistently maximizes the function with approximately one tenth the number of evaluations required by the second-best algorithm (SA). The time taken by MIMIC is indeed higher, but the speed at which it attains the optimal value is significant, as can bee seen from the plot.

Conclusions

As we saw above, there are some problem domains which are more suited to certain randomized optimization algorithms. It would be instructive to see if it is possible to theoretically characterize these domains. The above experiments demonstrate that before choosing a randomized optimization algorithm to solve a particular problem, it is useful to have knowledge of the domain and the structure of the solution space to decide which algorithm would have a higher chance of success.

References

[1] Mohammad, Rami, McCluskey, T.L. and Thabtah, Fadi (2012) An Assessment of Features Related to Phishing Websites using an Automated Technique. In: International Conferece For Internet Technology And Secured Transactions. ICITST 2012 . IEEE, London, UK, pp. 492-497. ISBN 978-1-4673-5325-0

[2] Mohammad, Rami, Thabtah, Fadi Abdeljaber and McCluskey, T.L. (2014) Predicting phishing websites based on self-structuring neural network. Neural Computing and Applications, 25 (2). pp. 443-458. ISSN 0941-0643

[3] Mohammad, Rami, McCluskey, T.L. and Thabtah, Fadi Abdeljaber (2014) Intelligent Rule based Phishing Websites Classification. IET Information Security, 8 (3). pp. 153-160. ISSN 1751-8709

[4] Dataset obtained from UCI and OpenML Repository https://github.com/pushkar/ABAGAIL:

- Phishing Websites Data – UCI Repository, OpenML

[5] ABAGAIL Github repository