RANDOMIZED OPTIMIZATION

David Gong
Department of Computer Science
Georgia Institute of Technology, davidcgong@gatech.edu

March 4, 2019

ABSTRACT

This report seeks to explore four different local random search algorithms - Randomized Hill Climbing, Simulated Annealing, Genetic Algorithms, and MIMIC. Implementations were done in Java using the ABAGAIL library.

1 Introduction

A prior paper with regards to supervised learning sought to explore the performances and accuracy results of 5 different SL machine learning algorithms (Decision Trees, Neural Networks, Boosting, Support Vector Machine, and k-Nearest Neightbors). This time, however, the goal is to construct and optimize the best weights for neural networks in general. In this case, the Spambase data set from the UCI Machine Learning Repository will be used to conduct analysis.

For the randomized optimization tests for the dataset, the neural networks were set up with 20 hidden nodes to compromise between performance and problem complexity, and 25,000 iterations to measure the function of growing accuracy, which would be expected to slow down as more iterations are run.

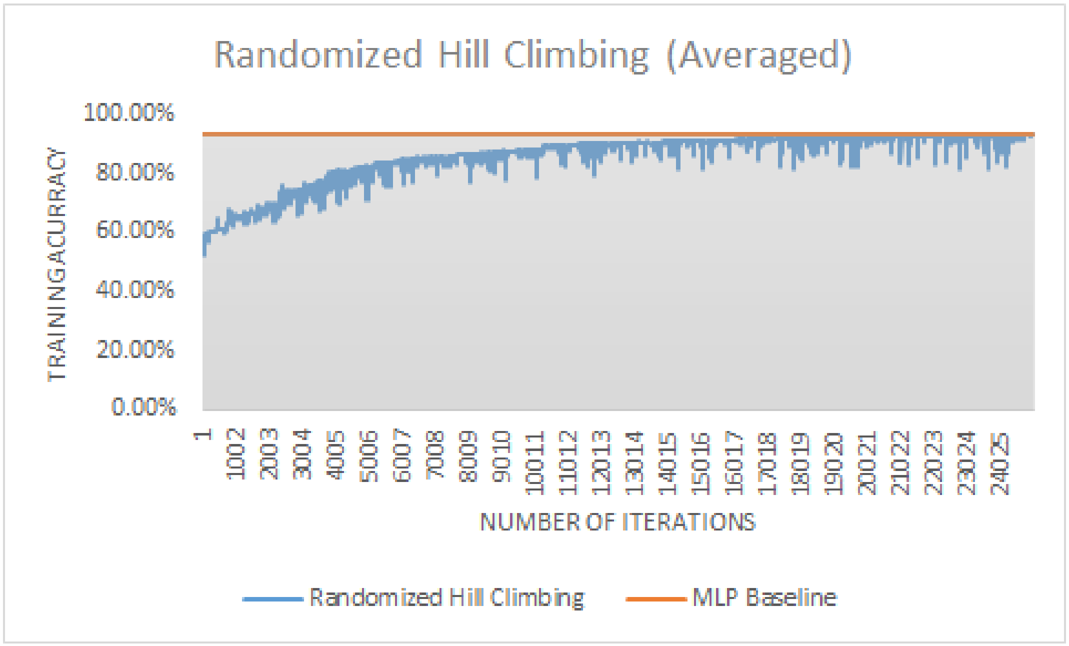
In addition, it should be noted that 'randomized' optimization itself suggests that performance and accuracy can greatly vary based on luck. As a result, often times multiple trials will need to be conducted in order to eliminate coincidence.

As a baseline to compare performance and accuracy, the previous results from the Supervised Learning assignment will be referred to. The Multi-Layer Perceptron neural network trained by backpropagation, which was adopted previously performed with 93.48% accuracy and took 26.97s to train.

2 Randomized Hill Climbing

Through the use of randomized hill climbing, once a random point is selected within the data, the algorithm makes incremental changes to the point in order to attempt to find the points of best fit. This continues until either the algorithm stops, or until the algorithm fails to improve at a point. This is where the ideal accuracy will be measured.

Due to the nature of the algorithm, it is important to perform multiple trials for the algorithm. Since a random point is selected using randomized hill climbing, there will be mixed results depending on how "lucky" the algorithm was when it came to selecting a point. Due to time constraints, however, only two trials of randomized hill climbing was performed. The graph that averages the two iteration results can be visualized as follows:



Though randomized hill climbing did eventually reach the baseline, the training time took 1053.17s on average, as opposed to 26.97s. In addition, randomized hill climbing under-performed on average, returning a 93.06% accuracy rate which is slightly less than that of the MLP neural network used for the SL assignment. The Gaussian distribution of the data also resulted in occasional spikes of error within the data, though this happens once around every couple of hundred iterations on average.

Though, one key point to make is that there were less hidden layers in the MLP neural network previously used, due to using scikit-learn's default configurations. This resulted in an increase in performance, and also surprisingly did not sacrifice accuracy.

However, when using ABAGAIL, a reduction to 5 hidden nodes while keeping 25,000 iterations halved the training time to 577.18s, but only resulted in an accuracy of 88.437%. Thus, the reason stands for having at least 20 hidden nodes to compromise between accuracy and performance.

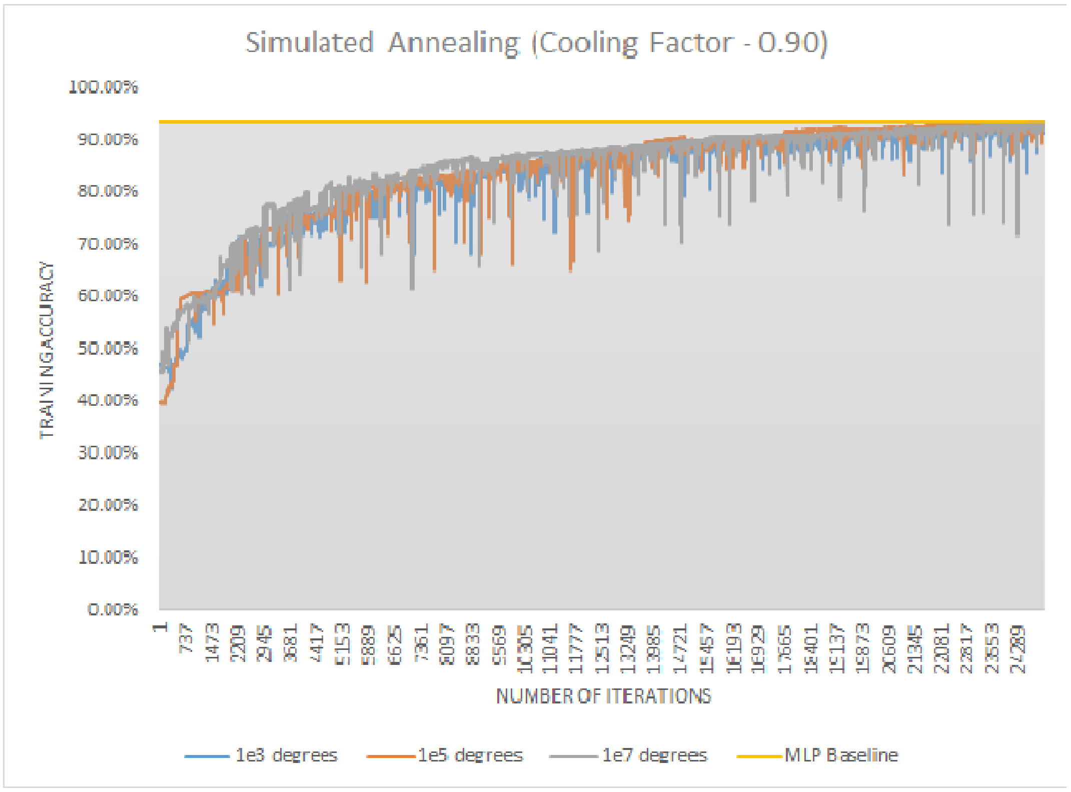
The logarithmic relationship between the number of iterations and training accuracy also shows that as the number of iterations increase, the training accuracy decelerates in terms of growth. This could be attributed to the decreasing likelihood of finding a better solution, as there are less better solutions as the training accuracy increases. As a result, to achieve even slightly better results as iterations increase would result in an exponential increase in time, if the case remains that there is still improvements to be made for the algorithm.

In conclusion, for this specific problem, randomized hill climbing was able to match up with the back-propagated neural network from the previous SL assignment, but took multiplicatively more time to do so. We look at simulated annealing next.

3 Simulated Annealing

For this problem, simulated annealing presents the possibility of approximating the global optimum. This differs much from gradient descent, which attempts to find the local optimum within the data. Surprisingly enough, the method of annealing hails from metallurgy, resulting in the usage of 'starting temperatures' and 'cooling factors' for this problem. Three sets of starting temperatures and cooling factors were examined:

Starting Tempatures {1e3, 1e5, 1e7}
Cooling Factors: {0.85, 0.90, 0.95}



As a standard to go by, a contrast was made between the trials for different starting temperatures (1e3, 1e5, 1e7) with the same cooling factor (0.90) in order to compare performance and accuracy. One thing to note is that due to the striking similarities between random hill climbing and simulated annealing when it comes to going through multiple iterations, multiple trials were done and averaged.

The following was the result for a cooling factor with 0.90:

Data with Cooling Factor - 0.90		
Starting Temperature	Training Time	Training Accuracy
1e3	1089.58s	93.65%
1e5	1064.33s	91.99%
1e7	1013.33s	92.98%

As can be seen, the training times and accuracy rates do not differ much when there is a change in starting temperature. In addition, the starting points for training accuracy as revealed by the graph should not be considered, but rather the plateau that the training accuracy reaches should be focused on. This is due to the fact that point selection is done at random, but convergence in general should be similar.

Additional tests with different cooling factors and starting temperatures show that even with varying combinations, there seems to be little to no correlation between these two factors with regards to training time and training accuracy. All of these iterations had one thing in common - high similarity in terms of training time and accuracy with randomized hill climbing.

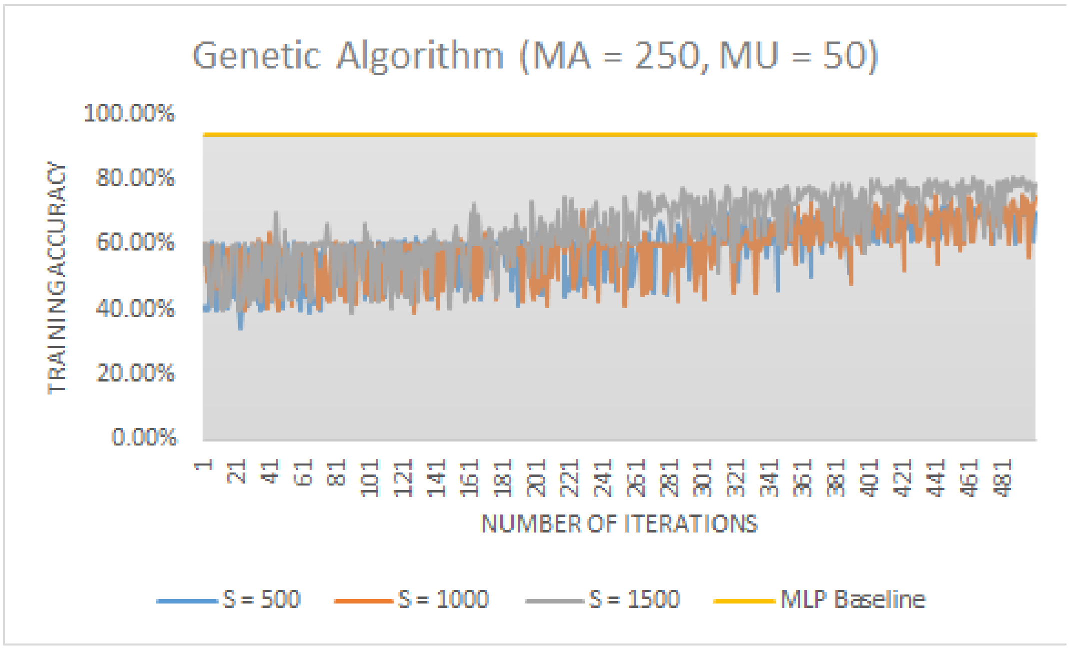
One thing to note, however, despite the similarities in results, is the amount of time it took for the differing starting temperatures to start to converge. In the beginning, starting temperature of 1e7 degrees seemed to converge much more quickly in the beginning than 1e3 and 1e5 degrees. The same can be said when comparing 1e5 to 1e3 degrees. As a result, it can be somewhat inferred that there is a correlation between higher starting temperatures and higher acceleration rate towards convergence. The behavior of such a phenomenon during the early iteration stages can likely be explained by the cooling factor itself, which increases the cooling off of the temperature. Eventually, the temperatures start to become similar in value, resulting in the similar convergence behaviors observable in the above graph.

Had simulated annealing been performed on a dataset that was at risk of converging to a local maxima, there would be differences in performance between the two randomized optimization techniques. This is due to the fact that randomized hill climbing is subject to being trapped by local maxima, whereas simulated annealing focuses on the global maxima.

4 Genetic Algorithms

The general approach of genetic algorithms was adopted as an optimization technique and was inspired by the idea of natural selection from biology, namely survival of the fittest. Using this idea, we observe the following when it came to applying this to the Spambase dataset.

Through varying the sample population size while keeping the number of mates per iteration and number of mutations per generation uniform, the following graph is presented:



Of immediate notice is the higher convergence value based on the sample population size. For this case, as sample population size increased, so did the convergence value and overall training accuracy. Upon viewing the table below, a few more details are revealed.

S = Sample Population
MA = Mating Population per Generation
MU = Mutations per Generation

Data with MA = 250, MU = 50		
Sample Population	Training Time	Training Accuracy
S = 500	1582.86s	73.44%
S = 1000	1667.98s	75.68%
S = 1500	1617.25s	80.461%

Based on the results shown, it seems that changing the sample population led to an improvement in accuracy while maintaining a similar training time.

The similar training time was an unexpected statistic - typically genetic algorithms would need to take longer to 'weed out' unfit genes from the data pool. However, in this case, the training time stayed relatively the same. This could be attributed to the relatively high ratio for MA and MU, which if set high or low enough would result in a multiplicatively difference in training time.

Though, in contrast to the previous two algorithms, however, genetic algorithms fall short in terms of both accuracy and training time. For this specific problem, Spambase had a large number of attributes/features, resulting in a relatively large search space. Though aforementioned is not the main issue, this combined with an early elimination of genes that were unfit led to a convergence towards the local maxima within the data. This would likely explain why a larger sample population size would lead to a better result, as it decreases the likelihood of converging to a local maxima within the dataset.

5 Additional Optimization Problems

In addition to applying the randomized optimization techniques to the Spambase dataset, three other general optimization problems were also researched to find the relative strengths of each randomized optimization techniques in different, general situations. All searching techniques used 25 trials in order to eliminate potential coincidence from random search/selection.

RHC = Randomized Hill Climbing, I = Iterations
SA = Simulated Annealing, T = Starting Temperature, C = Cooling Factor
GA = Genetic Algorithm, S = Sample Population, MA = Mating Population, MU = Mutations

5.1 Travelling Salesman Problem (Genetic Algorithm Advantage)

This well-studied problem is asked in theoretical computer science and the operations research domain. It presents the problem as: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?" In simpler terms, to solve this question, one needs to find the shortest and most optimal path to visit all nodes in a graph with N vertices.

For this problem specifically, N = 30 was used. The performance and accuracy of all the randomized optimization algorithms for this problem can be visualized below:

Travelling Salesman Problem Comparisons			
Search Technique	Parameters	Training Result (Averaged)	Total Time (25 Trials)
RHC	I = 25000	0.1613	0.84s
SA	I = 25000, T = 1e5, C = 0.90	0.1506	1.37s
GA	I = 750, S = 1000, MA = 250, MU = 50	0.1837	4.79s
MIMIC	I = 750, S = 1000, K = 250	0.1984	79.56s

Of immediate notice is MIMIC's better training result than GA on average, though that statistic will be elaborated on in following analysis.

Genetic algorithms work for this problem primarily because it looks at all the possible permutations throughout its iterations. Through filtering out 'unfit genes' and modifying children slightly from each generation, the genetic algorithm is able to make general improvements across the board. Seemingly one-dimensional improvements by RHC and SA is not sufficient for this problem. In addition, throughout every trial, there were no significantly differing values for genetic algorithms, signalling that genetic algorithms were consistently converging as opposed to fluctuating as can be seen in RHC, SA and MIMIC.

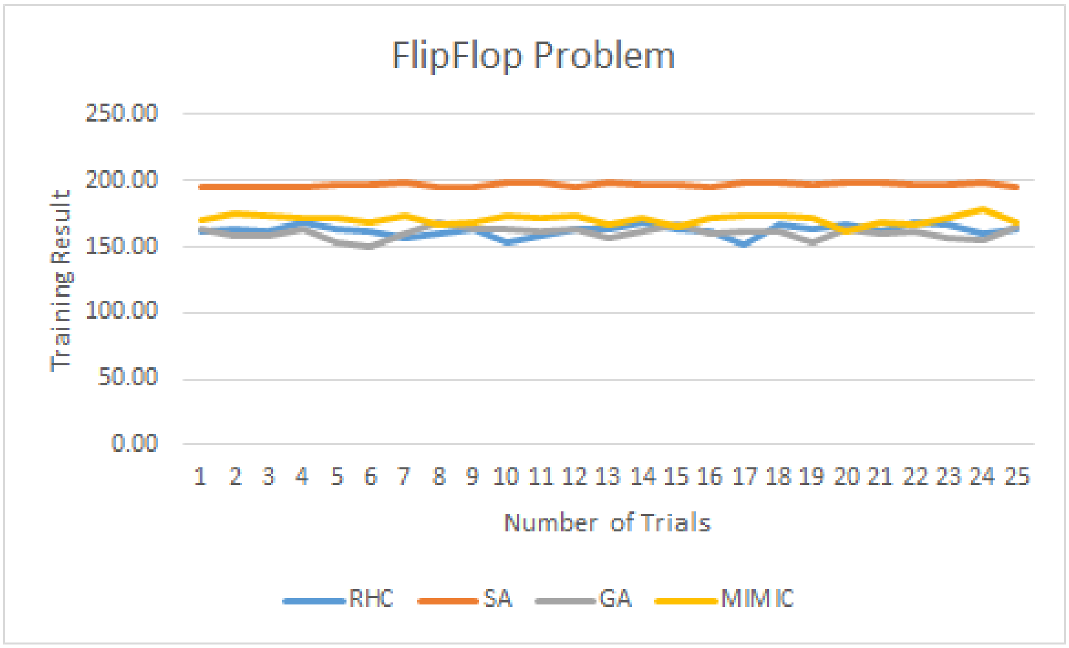
Though MIMIC did perform slightly better than GA in this instance, which is surprising in of itself, the time it took to perform each trail was multiplicatively more in contrast with the other searching techniques. MIMIC also fluctuates quite frequently from trial to trial, so its inconsistency is noted for this particular problem. This is most likely due to the high quantity of local maximas and large search space. This led to the low performance of MIMIC and inconsistency of training results.

After changing N = 30 to N = 50, there was a drastic decrease in performance for MIMIC, in addition with even exponentially longer training times. In fact, when N = 50, MIMIC became the worst performing search technique out of all the other techniques, instead of being the best in terms of accuracy. This was due to the exponential increase in the search space, something that MIMIC simply does not perform well with in terms of accuracy and performance. As a result, we conclude here that GA is the ultimate winner here overall.

5.2 FlipFlop Problem (Simulated Annealing Advantage)

This problem is described as a function that returns the number of bits that alternate within a bitstring. In order to obtain an optimal solution, a constantly alternating bitstring that returns a value equal to the length of the string would need to be returned. Tests were conducted on bitstrings of length 200.

The performance and accuracy of all the randomized optimization algorithms for the FlipFlop problem can be visualized below:



FlipFlop Problem Comparisons			
Search Technique	Parameters	Training Result (Averaged)	Total Time (25 Trials)
RHC	I = 25000	162.44	2.129s
SA	I = 25000, T = 1e4, C = 0.925	197.04	2.211s
GA	I = 750, S = 1000, MA = 250, MU = 50	160.48	2.171s
MIMIC	I = 750, S = 1000, K = 250	170.84	558.370s

Of immediate notice is that SA performed the best in terms of both accuracy and performance.

When it came to finding the most optimized answer, simulated annealing outshined the other searching techniques by a large margin. Performance was also much faster than that of MIMIC and similar to that of RHC and GA.

Upon further analysis of the problem, it can be observed that there is quite a large amount of local maximas for this specific problem. RHC can often be subject to converging to the local maxima it finds, which in this case can be quite troublesome. SA on the other hand tries to converge to the global maxima, and in this specific scenario, that would be the key to finding an optimal solution.

Simply put, being able to climb out of these convergence zones is the reason why SA was the best searching technique for this problem.

5.3 Four Peaks Problem (MIMIC Advantage)

This problem describes a function that takes a bitstring of length N and a trigger position T. An optimal solution would involve maximizing at bitstrings that have contiguous values of 0 and 1 within the range of the trigger point, and then the complementary bit from the trigger point to the end of the bitstring. Tests were conducted with parameters of N = 100 and T = 8 for this problem.

The performance and accuracy of all the randomized optimization algorithms for the Four Peaks problem can be visualized below.

Four Peaks Problem Comparisons			
Search Technique	Parameters	Optimal Result Percentage	Total Time (25 Trials)
RHC	I = 25000	0%	1.100s
SA	I = 25000, T = 1e4, C = 0.925	52%	1.363s
GA	I = 750, S = 1000, MA = 250, MU = 50	N/A	2.463s
MIMIC	I = 750, S = 1000, K = 50	72%	70.106s

MIMIC once again took the longest as expected to perform its set of trials, but greatly outperformed the other searching techniques in terms of finding optimal results.

RHC did not perform up to speed mostly due to the multiple amounts of local minima. This resulted in a failure to converge or take into account for other local minima, which ultimately resulted in a suboptimal solution every single iteration. SA did slightly better mostly because it is more focused on trying to find the global maxima, but as the multiple local maximas are neatly partitioned for the four peaks problem, SA would still not be able to produce 'optimal' results. Lastly, GA was the most intriguing technique for this problem, as it neither converged to an optimal solution nor a suboptimal solution. More iterations would perhaps be needed, but overall it simply proves that GA is not an optimal solution to this problem.

The reason why MIMIC does well for this specific problem can most likely be attributed to its tendency to find local optima by estimating probability densities. For this specific problem, the peaks are seemingly partitioned in a very similar manner and maintain similar distances to each other. As a result, MIMIC is able to pinpoint more precisely where these local optimas are located and account for them.

Through repetition and sampling of previous iterations, MIMIC will eventually be able to converge to the absolute maxima. As a result, it does well for this problem in particular. One thing to keep in mind, however, is that MIMIC is still rather lacking in terms of performance.

In addition, MIMIC generally does not perform well when it comes to Gaussian distribution, mostly because an estimation of the probability density in that manner is sure to be overshoot or undershoot the location of said optima.

Overall, due to the probability density of the Four Peaks problem, MIMIC is able to perform extremely well in contrast to the other search techniques which are prone to converging to local optimas or would need more iterations.