Randomized Optimization

Introduction and Data Description

In this assignment, I used the JAVA ABAGAIL library to explore four different randomized optimization algorithms: randomized hill climbing, simulated annealing, genetic algorithms, and MIMIC. Part I of this paper discusses the use of randomized hill climbing, simulated annealing, and genetic algorithms to find optimal weights of the neural networks for the UCI Credit Card dataset, while Part II reports on three different optimization problems used to evaluate all four of the optimization algorithms mentioned. The UCI Credit Card data set came from Kaggle and includes a variety of attributes such as default payments, demographic factors, credit data, history of payments, and bill statements. The data set includes a total of 30,000 instances and 24 variables. For the purposes of this assignment and allowing for reasonable runtimes, I created a random sample data set of 5,000 rows.

I updated my Neural Network from Assignment 1 based on the feedback in order to determine the best parameters from my backpropagation algorithm. I used scaled data with a 10-fold CV and 15 different sample sizes. I used a relu default activation and focused on looking at the different accuracy value for different numbers of hidden layers. I tested single layers with small sizes as well as double and triple layers of higher size since too many hidden layers cause high variance and not enough cause high bias. Figure 1 shows the graph of the different combinations of hidden layers. The model performed best with 1 hidden layer as the testing accuracy did not increase significantly with more layers, but if anything, it decreased. I used this to inform my hidden layers parameter for running randomized hill climbing, simulated annealing, and genetic algorithms in Part I. Retraining the backpropagation model with these hyperparameters provided a model accuracy of 0.81 and an AUC of 0.73. I also chose to use 5,000 iterations as my baseline for the randomized optimization algorithms tested in this assignment based on the results of my backpropagation neural network for comparison.

Figure 1

Part I

In this section, I will discuss the results from the application of three different randomized optimization algorithms: randomized hill climbing, simulated annealing, and genetic

algorithms. I will use the hyperparameters that provided the best results from my backpropagation algorithm. Each of the three algorithms were run using 1hidden layer with 5,000 iterations.

Randomized Hill Climbing

Randomized Hill Climbing (RHC) is a local search algorithm that that begins with a random point and searching for the local optima by moving towards neighbors that have more optimum values until it reaches a peak or optima. Starting with a random point increases the chance of finding a global optimum rather than being stuck at a local optimum.

The RHC algorithm I ran correctly classified 3,868 instances and incorrectly classified 1,132 instances to give an overall percent of correct classification of 77.360%. The training time of the RHC was rather short in comparison to the other two methods discussed in the upcoming sections with 2726.478 seconds and a testing time of 0.045 seconds. These results are summarized below in Figure 2.

Results for Random Hill Climbing	
Correctly Classified	3868 instances
Incorrectly Classified	1132 instances
Percent Correctly Classified	77.360%
Training Time	2726.478 seconds
Testing Time	0.045 seconds

Figure 2

The outputted sum of squared error results from the run by iteration are shown in Figure 3. This graph indicates that the number of iterations needed to converge is quite low for this data set as it seems to converge far before the 5,000 iterations.

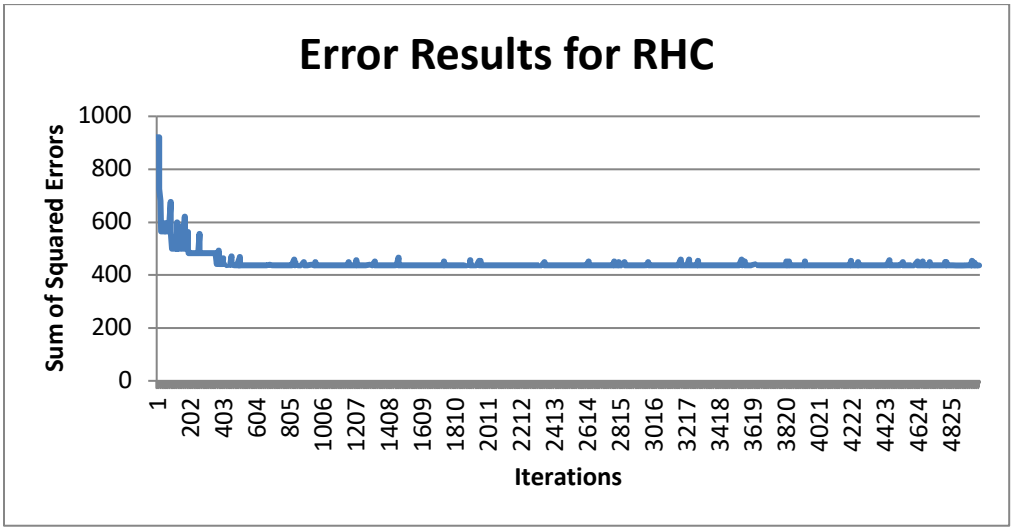

Figure 3

Simulated Annealing

Simulated Annealing (SA) also is a search algorithm that evaluates whether the new neighbor point is better than the previous in order to eventually find a global optimum. The algorithm comes from the process of finding the best way to determine how ductile metal can be through heating and cooling to change the shape of the metal itself. Like the RHC algorithm, it uses the same technique of looking to neighbors to find increased performance. If the current point is better than its neighbor a process based on the acceptance function is used to determine the probability of moving to this new point. Based on this probability, a worse point may be accepted. Higher temperatures provide more of an opportunity to move to new points versus cooler temperatures when looking to accept worse function values. This process allows for the finding of a global optima. The process is more thorough, but thus it is slower. Below are the results for SA algorithm run on the UCI Credit Card data set shown in Figure 4 and Figure 5.

Results for Simulated Annealing					
	0.15	0.35	0.55	0.75	0.95
Correctly Classified Instances	3875	3875	3875	3875	3875
Incorrectly Classified Instances	1125	1125	1125	1125	1125
Percent Correctly Classified	77.50%	77.50%	77.50%	77.50%	77.50%
Training Time (seconds)	273.867	258.575	258.479	2405.049	8596.206
Testing Time (seconds)	0.025	0.014	0.012	0.010	0.012

Figure 4

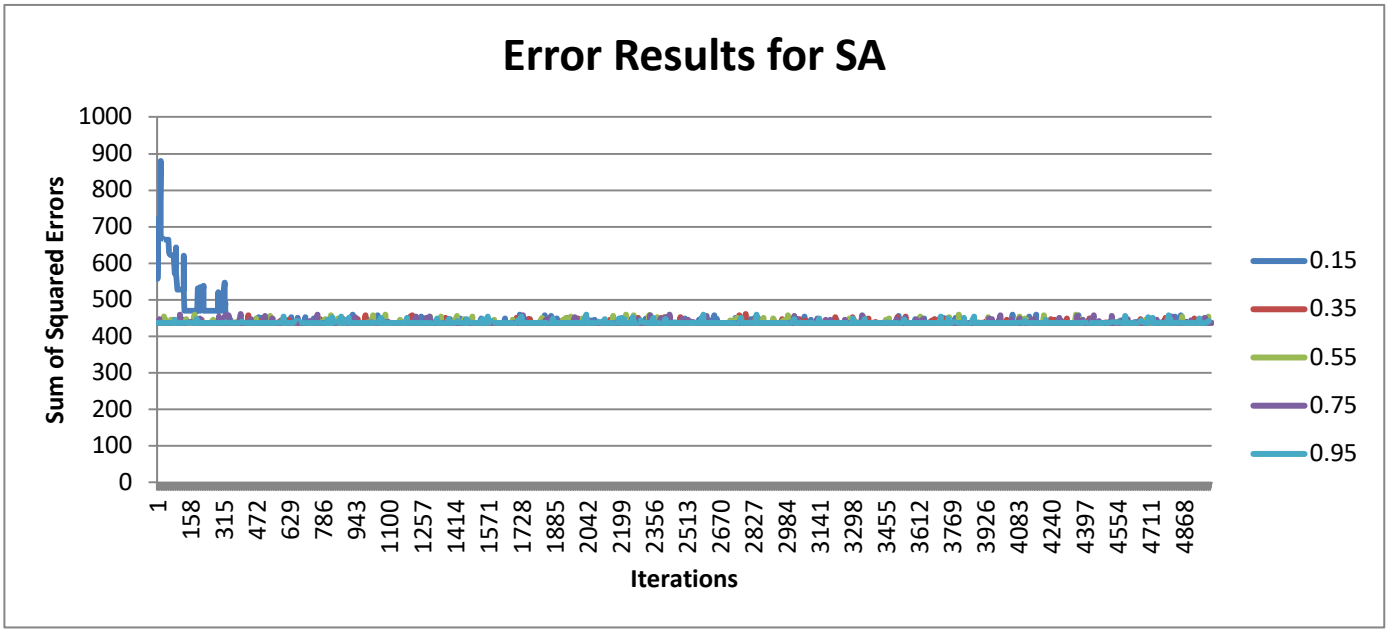

Figure 5

Figure 5 shows the sum of squared errors or error rate results for the SA test for each iteration performed (5,000). The model seems to converge well before the 5,000 iterations, which was also the case for RHC. This would indicate that the model may just not need that many iterations to perform best. Given the fact that the accuracy performed the same for all cooling rates I did not provide a graph for this. The graph for training times by cooling rate for 5,000 iterations is shown in Figure 6.

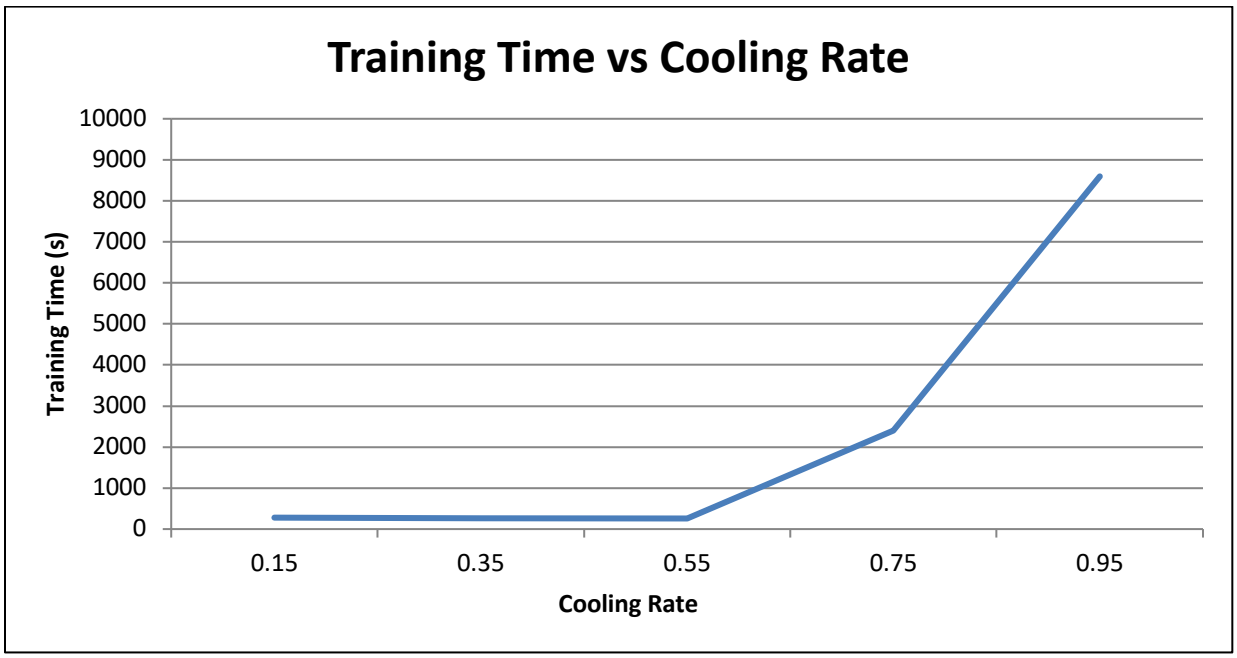

Figure 6

This graph of training times versus the cooling rate shows that the training time increased significantly from 0.55 to 0.95 after initially being quite low for the first three cooling rates. From these graphs and given the fact that the accuracy is the same for all cooling rates, the 0.55 cooling rate performed the best in minimized the sum of squared errors while still maintaining a relatively low training time.

Genetic Algorithm

Genetic Algorithm (GA) is a method for solving constrained and unconstrained optimization problems. It is based on the biological process of natural selection, specifically the process that drives biological evolution by finding the fittest options to survive and produce offspring. This selection of parents is initially done at random and they produce children for the next iteration of solutions. Eventually the process will find the optimal solution as the population evolves.

The training time for GA was by far the longest of the three algorithms and was initially taking well over 24 hours to run. As a result, I had to restart and decrease my number of combinations from 5 to 3. Figure 7 shows the results of these with the lower population size of (10, 5, 2) combination having the lowest sum of squared errors results over the 5,000 iterations and thus performing the best. As with RHC and SA, it seems as though the GA model too

converges much before the 5,000 iterations finish indicating not that many are needed for this data set.

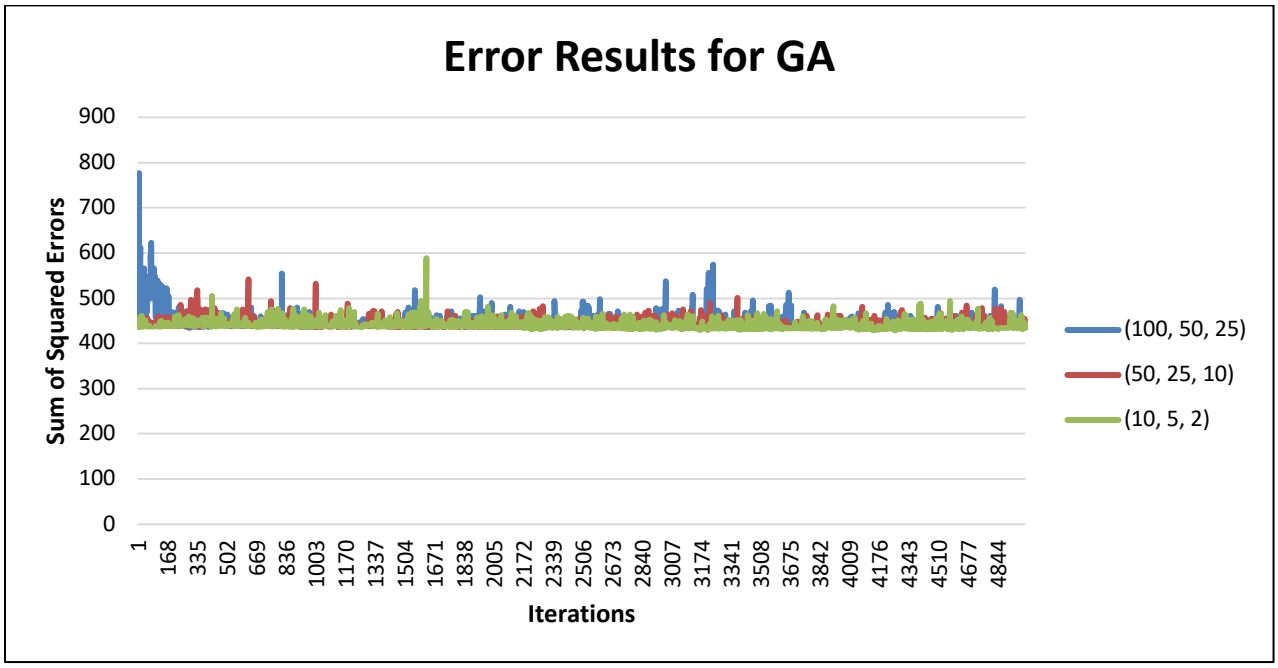

Figure 7

Also like the SA algorithm, the accuracy rates were the same for all three combinations at 77.36% making my choice of best performance more based on the lowest sum of squared errors.

Algorithm Comparison

In Figure 8, the test accuracy for the three new algorithms as well as the accuracy from the original Backpropagation algorithm are shown. It would seem that the Backpropagation algorithm performs better than the RHC, SA, and GA, with SA performing second best by only a slight margin from RHC and GA. GA and RHC had the same accuracy score. Given the significantly longer training times associated with RHC, SA, and GA, it would imply that Backpropagation would be the best algorithm to use amongst the four.

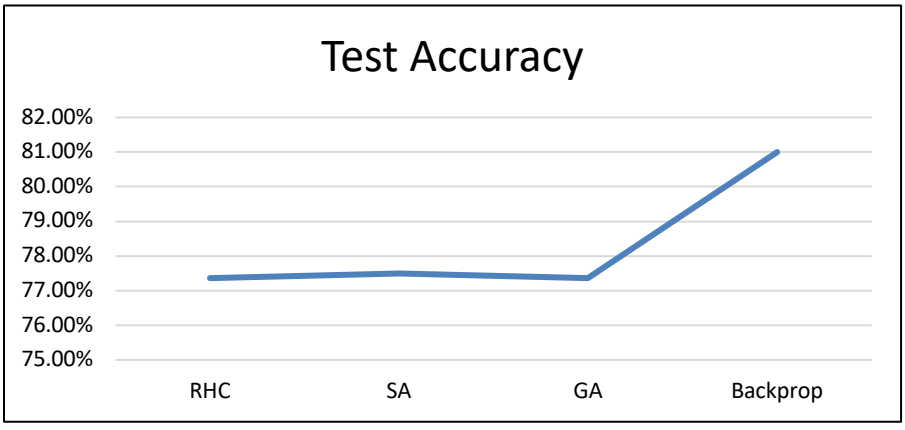

Figure 8

Part II

In this section, I chose three optimization problems to use on the four randomized optimization problems: randomized hill climbing, simulated annealing, genetic algorithms, and MIMIC. The three problems I chose are knapsack, continuous peaks, and traveling salesman. Using 5,000 iterations for RHC, SA, and GA (as I did in Part I) and 1,000 iterations for MIMIC (as was the default, and I determined to be suitable for runtime feasibility), I chose the parameters for each based on what performed best in Part I. Based off my Part I results, I chose the 0.55 cooling rate for the SA parameter and the 10, 5, 2 combination for the GA. For mimic, I choose the 200, 100 combination in order to make sure I was able to run them in a timely manner. The results for the three optimization problems follow in Figures 9-13. I also investigated further different parameter values for the optimization algorithms that performed best for the knapsack and continuous peaks problems (all still using 5,000 except MIMIC with 1,000).

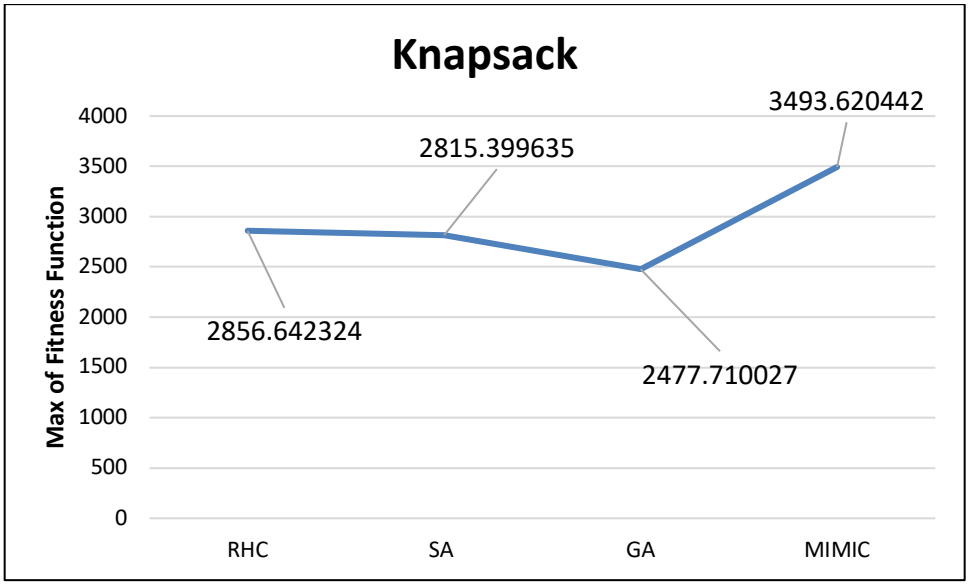

Figure 9

For the knapsack optimization problem, the MIMIC algorithm performed the best with a max fitness function value of 3,493.62. The difference in performance time was almost inconsequential, though MIMIC did take the longest. However, it's large increase in performance would suggest it works the best. The knapsack optimization problem determines how many weighted items in a collection to include so the total weight is under some determined threshold while producing the largest value possible.

These results would indicate that this problem would allow MIMIC to perform the best as it is a computationally expensive algorithm and the knapsack problem works well with those types of algorithms. Figure 10 is a graph that shows the different MIMIC samples for investigation.

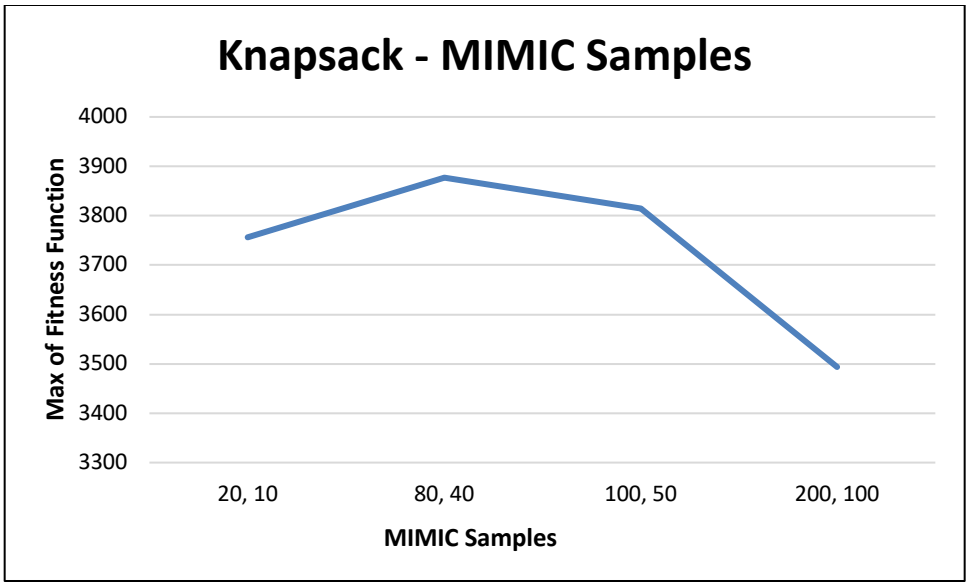

Figure 10

From this Figure 10 graph, it would seem that some sort of sample combination around 80,40 would perform the best, this might be due to that being a good sample combination for this particular data set.

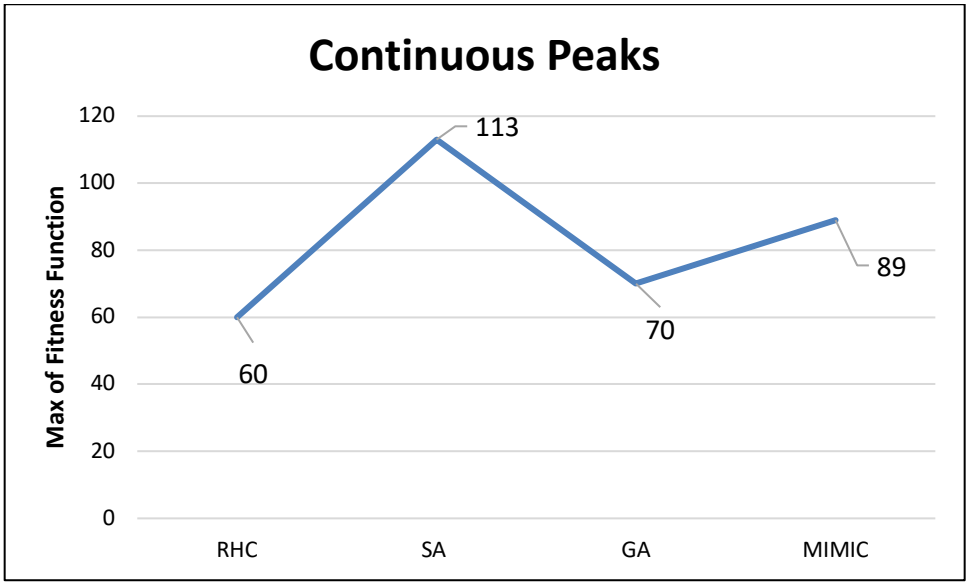

Figure 11

The continuous peaks optimization problem showed different results than knapsack, with simulated annealing performing the best with a max fitness function value of 113. Again, the runtimes were almost inconsequentially different, with MIMIC though taking the longest to output an optimal solution. This problem focuses on finding the highest peak as a global optima.

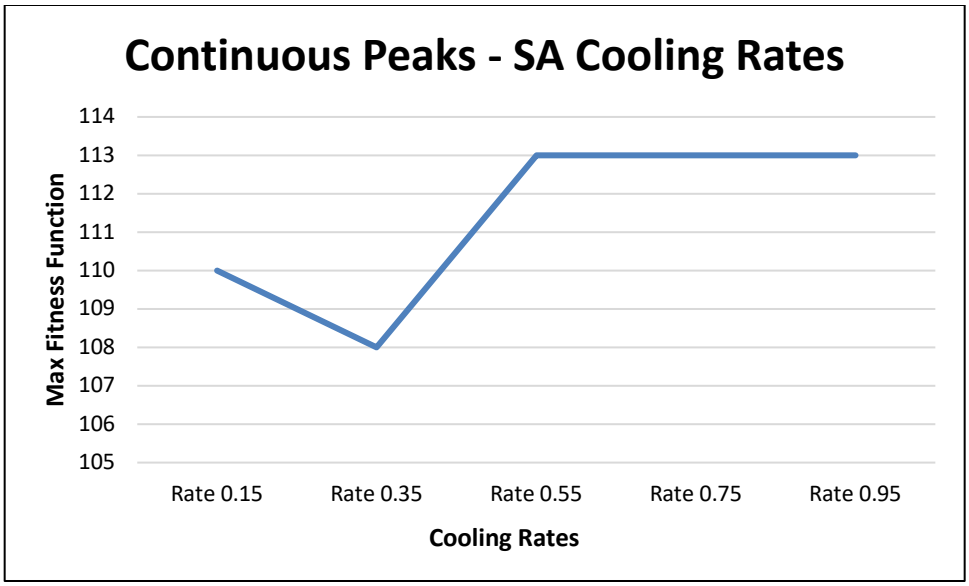

Figure 12

To further investigate the performance of SA using the continuous peaks problem, I tested the different cooling rates I used in Part I. From Figure 12 it can be seen that all rates still give a higher fitness function value than any of the other optimization algorithms did (RHC, GA, MIMIC). It would also seem to show that the max fitness function value may plateau around 0.55 and that a higher value cannot be achieved as the values for the 0.55, 0.75, and 0.95 cooling rates are all the same. This could indicate that an optimum point can be achieved without having to maximize cooling rate to search more aggressively for a better neighbor.

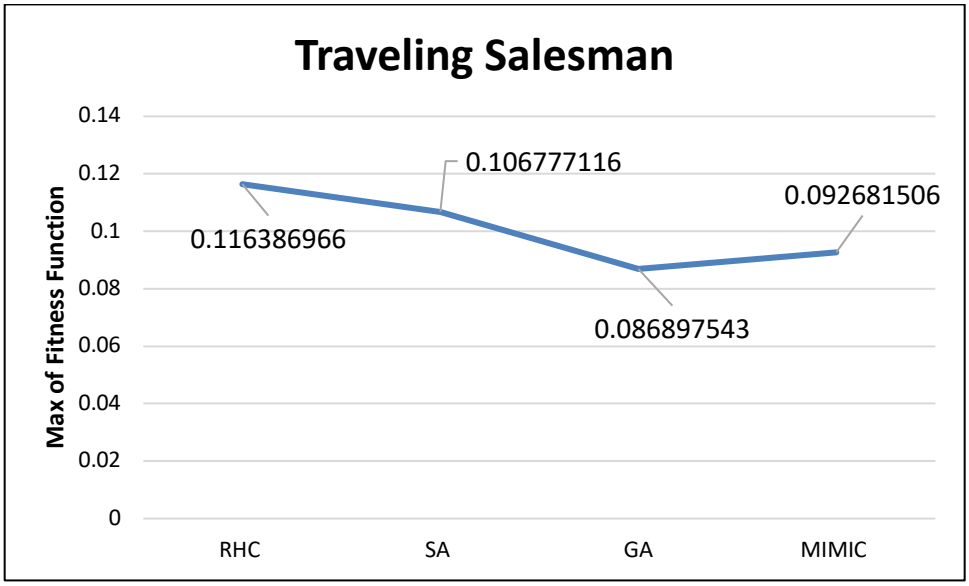

Figure 13

Finally, the traveling salesman optimization problem had the best performance with randomized hill climbing with a max fitness function value of 0.11638696. This problem focuses on finding the shortest path between different points. Given that there are no additional parameters to change for RHC, I did not investigate this further. However, it makes sense that RHC would perform the best with this problem since RHC is generally not as computationally expensive with respect to time.

In all three optimization problems, MIMIC took the longest, though none of the four algorithms took too much time to complete.

Conclusion

Overall, the optimization problems all perform differently with the different optimization algorithms and have pros and cons depending on whether you are looking for increased speed or performance. RHC and SA usually have the lowest computation times while MIMIC and GA are rather computationally expensive, though in this case all output within under 1 minute while performing the three different optimization problems in Part II. In Part I, RHC and SA had the fastest runtimes overall while GA was extremely long. This is most likely due to the nature of the problem and how it produces children for finding the optimal solution. I was able to decrease runtime by decreasing the number of combinations I used for my population, mating, and mutation processes. For SA, I could've tested more cooling rates to see if there was something more accurate than the 0.55 value I chose as the most reasonable for use in my Part II problems. As for RHC, given the fact that there are no additional parameters to change, iterations would have been the only potentially feasible option for trying to improve performance, other than maybe using a smaller dataset to decrease testing and training times (though in general these were already pretty low for this).

One thought I had in general to improve performance was to test different numbers of iterations. I used 5,000 for all for runtime feasibility and based on the learning curves of my original backpropagation neural network and what worked well with that (in terms of convergence). One reason I did not try more iterations was because for all of the algorithms, it seemed that they were converging far before the 5,000 indicating that perhaps not that many would be needed for this data set.

Ultimately, backpropagation was shown to still be superior on both runtime and accuracy in comparison to the three optimization algorithms investigated in Part I. Backpropagation was also able to handle running much more data to create a more robust neural network than the other three which I had to use samples of the original data set for in order to get reasonable run times. This all makes sense as backpropagation works by feeding information back to the network each iteration thus tuning the weights for continued improvement of classification accuracy based on how it's performing. This is not how the randomized optimization neural network algorithms work and thus they performed worse and took longer.