Joshua Demeo (gtid: jdemeo3)

Neural Network Optimization and Toy Randomized Optimization Problems

Abstract

In this work, a number of randomized optimization algorithms (Randomized Hill Climbing, Simulated Annealing, Genetic Algorithm) were applied to the neural network for the MAGIC Gamma Telescope[1] dataset from the supervised learning assignment. Additionally these 3 algorithms along with the MIMIC algorithm were applied to 3 optimization problems: Knapsack, Four-Peaks, and the Travelling Salesman problem.

Importance of the Datasets and Problems

The gamma dataset is interesting for the sake of understanding the differences between the randomized optimization algorithms when applied to Neural Networks. This is because the dataset is not able to be classified perfectly by the accuracy metric and differences between optimization algorithms are noticeable. Also the shape of the MAGIC Gamma Telescope (19020 rows, 10 features) dataset is interesting with its low feature count, which prevents overfitting to noise that may be apparent in cases where there are tons of features. The toy optimization problems are all interesting because each problem has significant performance differences between each of the optimization algorithms.

Methodology

All models were implemented with the ABAGAIL[2] library and were run using jython[3]. The process of testing and analyzing each optimization algorithm are as followed:

- Neural Network-Gamma dataset:
    - The datasets continuous features were normalized, it's samples were shuffled, and then split into a stratified 50/50 train/validation split.
    - The network architecture consists an input layer of 10 nodes, 2 hidden layers with 10 nodes each, and output layer of 1 node. The activation function between nodes is the ReLU function[4]. Sum of squared errors is the metric being optimized as ABAGAIL can implement it in binary classification tasks.
    - 2-fold cross validation with a gridsearch was used to tune hyperparameters for each algorithm.
    - Models were only evaluated using 2-fold cross validation and not with a final held out test set. This was to have more training data available for each optimization algorithm to see if more data would help them outperform Backpropagation.
    - The optimal value for the hyperparameter of number of iterations/weight updates was determined by where a plateau in performance was for each algorithm
- Optimization Problems
    - The optimization problems evaluated include: Knapsack, Four-Peaks, and the Travelling Salesman problem. Each was evaluated by the optimization

algorithms: Randomized Hill Climbing (RHC), Simulated Annealing (SA), Genetic Algorithm (GA), Mutual-Information-Maximizing Input Clustering (MIMIC).

- A small grid search of hyperparameters specific to SA, GA, and MIMIC was performed, but only marginal differences were observed so the default hyperparameters for the toy problems in ABAGAIL were used.
- The toy problems scenarios were not changed from their default parameters since the difficulty of each was sufficient to see differences between each optimization algorithm.
- Shorthand labels for **Figures**: RHC = Randomized Hill Climbing, SA = Simulated Annealing, GA = Genetic Algorithm, MIMIC = Mutual-Information-Maximizing Input Clustering

Results and Discussion

Neural Network Optimization for Gamma Dataset

Backpropagation

The backpropagation algorithm was applied to the neural network architecture from assignment one using the ABAGAIL library rather than Scikit-Learn to get a baseline model in this framework. The model reaches an optimal number of iterations (weight updates) after around 1000 iterations with a validation accuracy of 86.5%. This corresponds to a training time of 37.0 seconds (**Figure 1**). It is worth noting that significant overfitting isn't observed as shown by the increasing number of weight updates without significantly decreasing validation performance. This could possibly be described by how the distributions of the data points from the training and validation sets are very similar and help the network maintain even bias and variance.

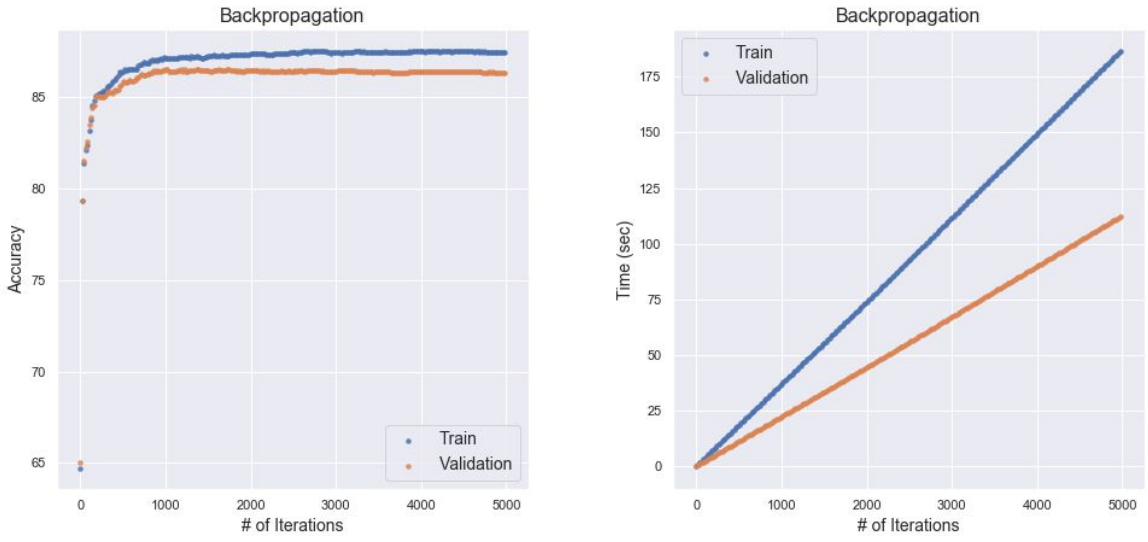

Figure 1. Backpropagation algorithm performance and time complexity versus number of weight updates.

Randomized Hill Climbing

The Randomized Hill Climbing algorithm had no hyperparameters beyond number of iterations to optimize so the default implementation was used. The model reaches an optimal number of iterations after around 4500 iterations with validation accuracy of 83.4% (**Figure 2**). This corresponds to a training time around 134.5 seconds. This performance is very close to that of the Backpropagation algorithm. This is surprising considering that Backpropagation can efficiently handle the continuous values of the weights via gradient descent, while Randomized Hill Climbing is checking nearest neighbors via discrete intervals. This could potentially result in overshooting some value that might be optimal between that interval.

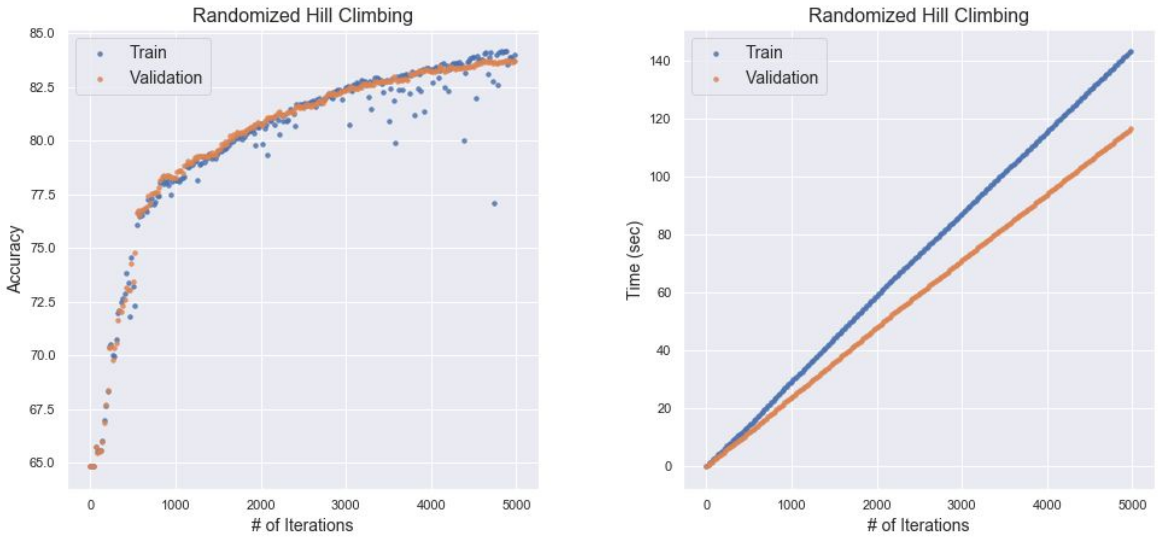

Figure 2. Randomized Hill Climbing performance and time complexity versus number of weight updates.

Simulated Annealing

The Simulated Annealing Climbing algorithm hyperparameter for rate of cooling was initially optimized, while the default temperature (1×10^{13}) was kept constant as it was sufficiently high to allow for gradual cooling. It was found that a lower cooling factor (lower factor reduces temperature value quicker over iterations) such as 0.8 corresponded to getting stuck in less optimal parameter space as the accuracy of the network suffered as compared to higher cooling factor values that were in the range of 0.90-0.98. The optimal cooling rate was determined to be 0.95.

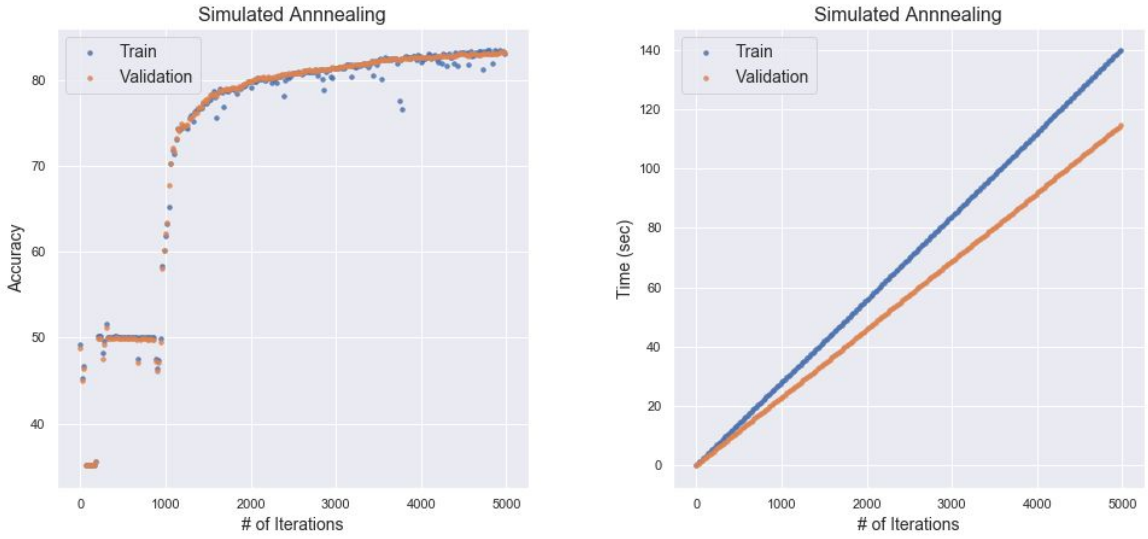

Figure 3. Simulated Annealing performance and time complexity versus number of weight updates.

This model reaches an optimal number of iterations after around 4000 iterations with validation accuracy of 82.5% (**Figure 3**). This corresponds to a training time around 124.2 seconds. This performance is very close to that of the Backpropagation algorithm and very similar to that of Randomized Hill Climbing. It isn't entirely surprising that the performance is similar to Randomized Hill Climbing as the algorithm performs more and more similarly to Randomized Hill Climbing as the temperature decreases with each iteration. The same issues as discussed in the Randomized Hill Climbing section with respect to weight updates over discrete intervals also applies to Simulated Annealing.

Genetic Algorithm

The Genetic Algorithm hyperparameters were initially optimized. It was found that higher population size corresponded to lower accuracy over the training and validation sets. It was found that a population of 150 was optimal, with the top 100 performing set of parameters being crossed over, and then the top 10 being mutated.

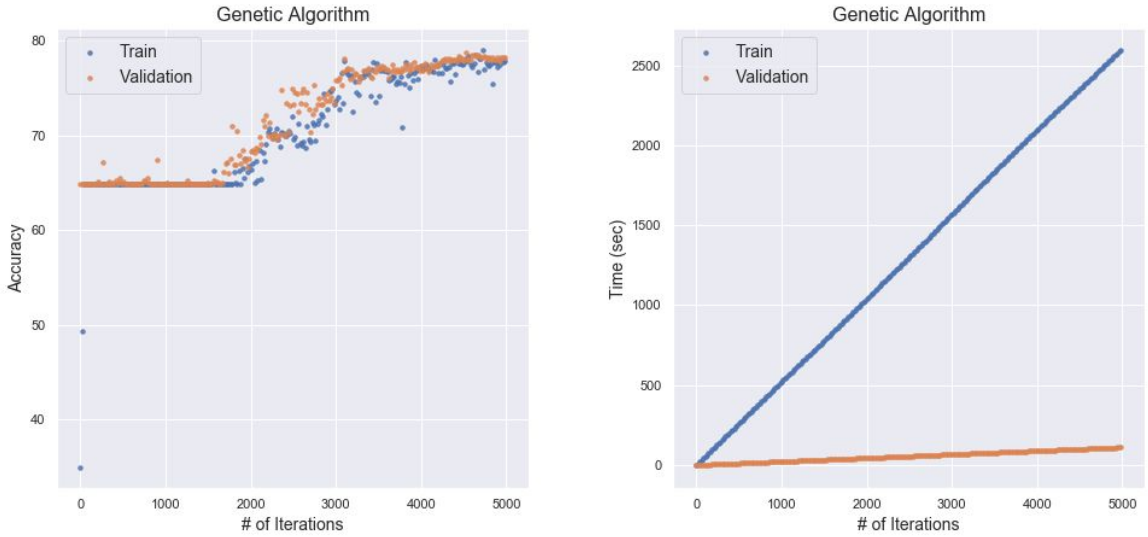

Figure 4. Genetic Algorithm performance and time complexity versus number of weight updates.

This model reaches an optimal number of iterations after around 3500 iterations with validation accuracy of 77.7% (**Figure 4**). This corresponds to a training time of 2371.9 seconds (~40 minutes). This performance is very poor with regards to accuracy and time as compared to the otherl optimization algorithms. The accuracy across both the training and test sets was not very smooth across the iterations, which possibly exemplify how cross-over can create very different hyperparameter spaces where the combination of two strong parents doesn't always correspond to a strong performing child.

Summary

Unsurprisingly Backpropagation performed the best on this optimization task by validation accuracy and training time to reach its optimal set of network weights (**Table 1**). This can be attributed to gradient descent capturing all real values. The instance based algorithms (RHC, SA) performed very similarly, while Genetic Algorithm performed poorly for this task.

Table 1. Summary of performances by optimization algorithms on Gamma dataset with 2-fold Cross Validation

Optimization Algorithm	Number of Iterations	Training Time (sec)	Train Accuracy	Validation Accuracy
Backpropagation	1000	37.0	87.2%	86.6%
Randomized Hill Climbing	4500	134.5	83.9%	83.4%
Simulated Annealing	4000	124.2	82.5%	82.5%
Genetic Algorithm	3500	2371.9	78.4%	77.7%

Toy Optimization Problems

Knapsack Problem

The Knapsack Problem is an optimization problem where a set number of items with a set number of weights and associated volumes must be added to a knapsack of a constant volume. The optimization task is to maximize the total weight (value) of all items combined that is less than the max volume of the knapsack. Analogous examples in the real world include having the lightest weight and maximally filled backpack for a backpacker or maximizing the amount of parts created during manufacturing with the least amount of unused base material.

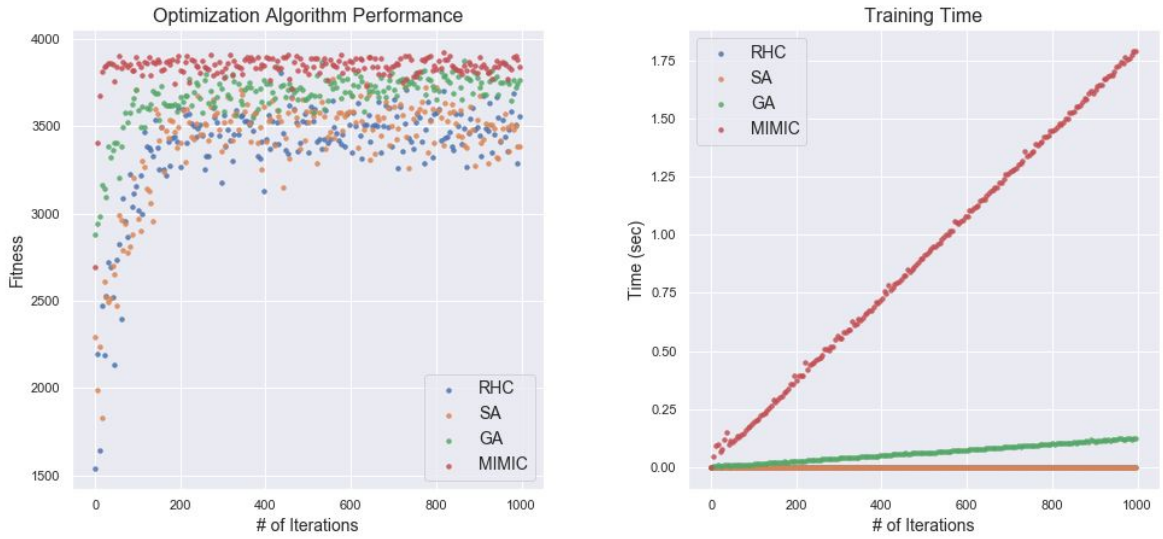

Figure 5. Optimization Algorithms performances and time complexity versus number of iterations for Knapsack Problem; *see text for definition of fitness

For this task, the MIMIC algorithm performed the best by fitness (maximal weight/value added to the knapsack) against all other optimization algorithms (**Figure 5**). Increasing the number of iterations beyond 1000 showed no difference in performance for each optimization algorithm (data not shown). The MIMIC algorithm was able to reach its optimal set of parameters relatively quickly in under 50 iterations, while the other 3 optimization algorithms reached their optimal set of parameters close to 200 iterations. There is a significant time penalty with MIMIC with increasing iterations, but because the optimal set of parameters were reached relatively quickly (< 0.25 seconds), the amount of time wouldn't be considered prohibitive of its use in this scenario. The MIMIC algorithm likely performs well because it is able to develop a probability distribution of the best set of parameters and won't be stuck as easily in local optima through such sampling. The instance based algorithms (RHC, SA) are unable to reach the same optima as MIMIC, which indicates that the optimization problem likely has many local optima. These algorithms are likely choosing the highest valued item even if it takes up

alot of volume, which makes it difficult for such greedy approaches to reach a potential global optima.

Four-Peaks Problem

The four peaks problem sets up an N-dimensional input vector of zeros and ones where there exists two global maxima and two local optimum for the fitness function. The fitness function looks at how many consecutive zeros there at the start of the input vector and how many consecutive ones are at the end of the input vector and returns the maximum of the two. In addition though, there exists a case where if the number of zeros and ones are greater than a specific threshold, a set number of points (usually 100) is added to the fitness function. This scenario is where one could reach one of the global optima.

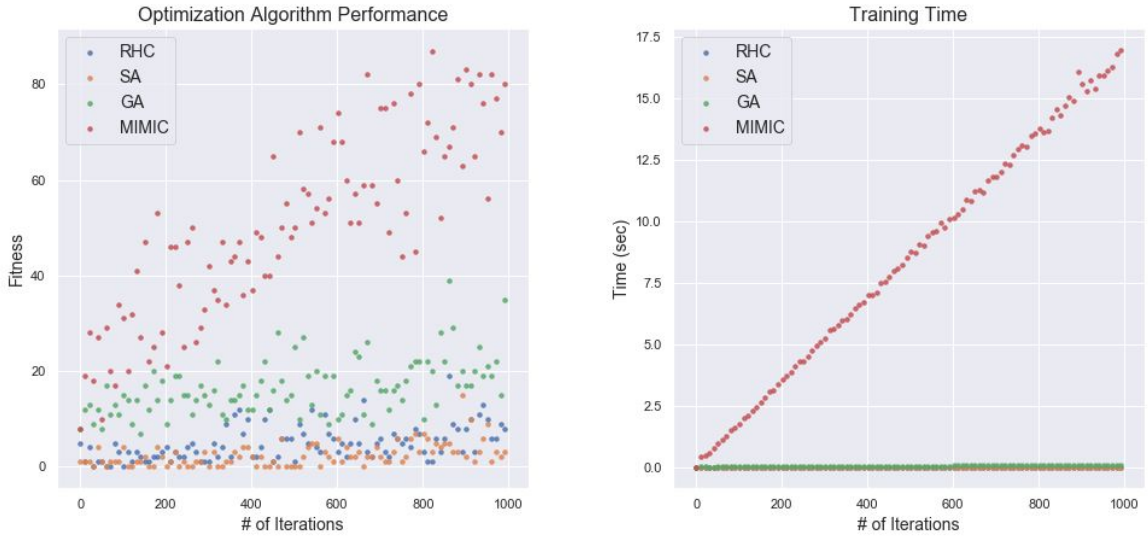

Figure 6. Optimization Algorithms performances and time complexity versus number of iterations for Four-Peaks Problem; *see text for definition of fitness

For this particular problem, initial testing with 1000 iterations indicated that MIMIC was performing best, but there was a significant time cost associated with its performance (~17.5 seconds) (**Figure 6**). Since the time cost for the other 3 algorithms was low, they were tested and allowed to run for 50000 iterations (**Figure 7**). The instance based algorithms (RHC and SA) performed the best on this particular problem with a low associated run time cost for training despite the higher number of iterations. Both algorithms started to max out their fitness after about 30000 iterations. The instance based algorithms are able to reach the global maxima because they can easily exhaust the search space and reach each of the optima that exist. For Genetic algorithm and MIMIC, each must reach some intermediate equilibria where exploration isn't rewarding enough and thus a local optima becomes sufficient.

Figure 7. RHC, SA, and GA performances and time complexity versus number of iterations for Four-Peaks Problem; *see text for definition of fitness

Travelling Salesman Problem

The Travelling Salesman Problem (TSP) is the task of finding the shortest possible route from an origin to every single city and then back to the origin. The list of cities and the distances between each city are given. This problem is important to applications of determining optimal routes for transport and shipping. The fitness metric for this problem is the inverse of the shortest distance traveled (1 / routes distance).

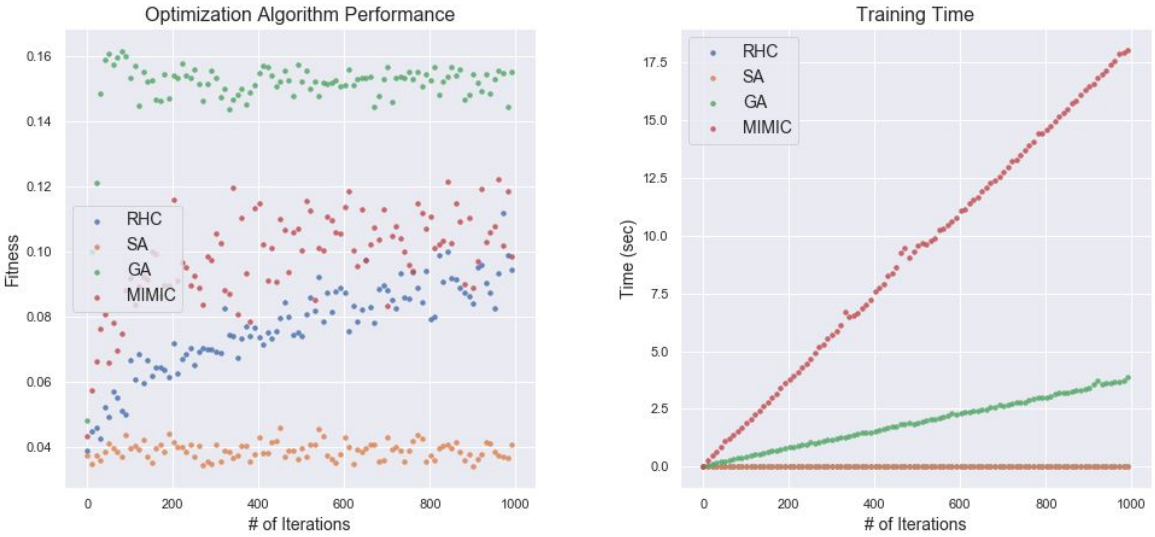

Figure 8. Optimization Algorithms performances and time complexity versus number of iterations for Travelling Salesman Problem; *see text for definition of fitness

For this particular problem, the Genetic Algorithm performed the best with an inverse distance traveled of around 0.16 in around 50 iterations (**Figure 8**). Increasing the number of iterations to 50000 showed some improvement in fitness for Simulated Annealing, but never reached the Genetic Algorithms performance despite more iterations (**Figure 9**). The MIMIC algorithm was tested with up to 5000 iterations but no obvious improvement in performance was observed (*data not shown). The search space for this problem is quite large (50! = 3.0 x 10^{64}) and a greedy search strategy (RHC, SA) would take far too many iterations to search this parameter space to reach the same optimum as the Genetic Algorithm. The Genetic Algorithm likely does well on this problem because it is able to search parameter space in an efficient manner. Crossover of high performing parameters are more likely to produce high performing offspring for GA because a subroute of each parent may be the most optimal for that set of cities and if the two subroutes are exclusive from each other, their concatenation may create a better route than each of the parent routes. Since crossover occurs for multiple individuals in the population for each iteration, this can increase the chances of exploring optimal regions of the parameter space.

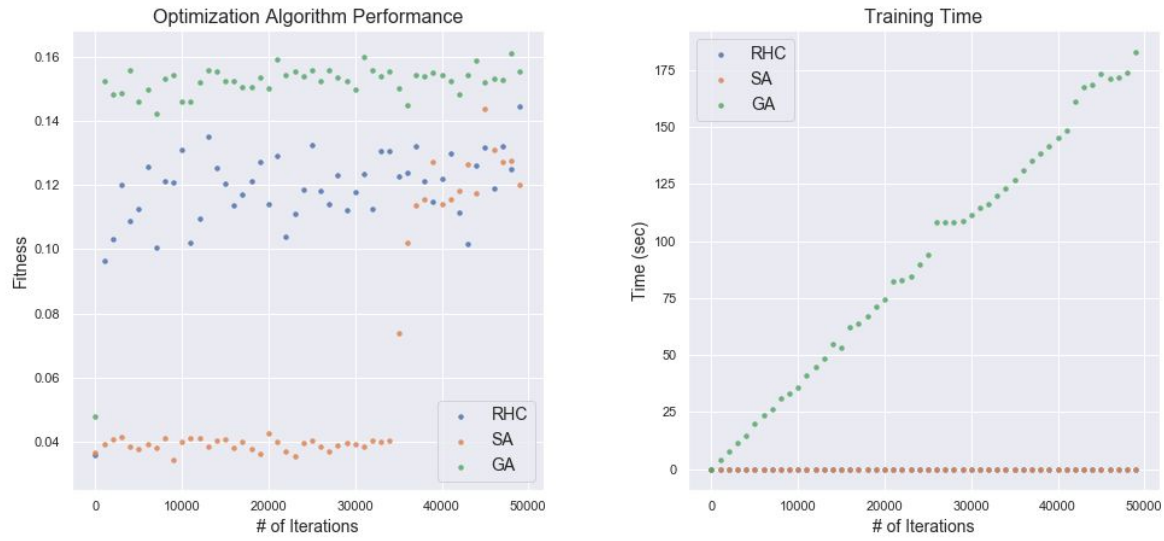

Figure 9. RHC, SA, and GA performances and time complexity versus number of iterations for Travelling Salesman Problem; *see text for definition of fitness

Conclusions

In this work, a number of randomized optimization algorithms (Randomized Hill Climbing, Simulated Annealing, Genetic Algorithm) were applied to the neural network for the MAGIC Gamma Telescope[1] from the supervised learning assignment. Additionally these 3 algorithms along with the MIMIC algorithm were applied to 3 optimization problems: Knapsack, Four-Peaks, and the Travelling Salesman problem. MIMIC was able to efficiently solve the Knapsack problem, while the Genetic Algorithm was able to efficiently search the large parameter space for the Traveling Salesman problem. The instance based algorithms (SA,

RHC) were able to find each of optima in the Four-Peaks problem and eventually reach the global optima for this toy problem.

References

1. MAGIC Gamma Telescope Data Set. UCI Machine Learning Repository. https://archive.ics.uci.edu/ml/datasets/magic+gamma+telescope
2. Java ABAGAIL Library. https://github.com/pushkar/ABAGAIL
3. Jython: Python for the Java Platform. http://www.jython.org/
4. ReLU implementation. https://github.com/JonathanTay/CS-7641-assignment-2/blob/master/ABAGAIL/src/func/nn/activation/RELU.java