Colby Tobin
Professor: Charles Isbell
March 3rd, 2019

CS4641: Randomized Optimization Analysis

Introduction

This paper is designed as an analysis and comparison of various randomized optimization algorithms. The paper is divided up into 2 main sections that are different, but relate in how the algorithms perform against each other. The first section deals with using the randomized optimization algorithms to determine the optimal weights of a neural net used to classify a provided dataset. In the case of this paper, optimal is defined as the neural net weights that result in the lowest error rates. These randomized optimization algorithms will be compared to the performance of backpropagation for determining weights. The second part of the paper will discuss comparisons between four optimization algorithms and their performance on 3 types of problems. The problems being optimized are the flip flop, max k-coloring, and knapsack problems.

The optimization algorithms being discussed in this paper are Random Hill Climbing (RHC), Simulated Algorithms (SA), Genetic Algorithms (GA), and MIMIC. The first section of the paper will use the RHC, SA, and GA algorithms. The second section of the paper compares all four algorithms.

Randomized hill climbing is a very simple random optimization algorithm. Given a point in space, the algorithm will look at points within the neighborhood of the original point, and pick the best point within that neighborhood as defined by a fitness function associated with the problem, and then travels to that point. This process loops until there is no movement meaning an optima or minima was found. Simulated Annealing is a very similar algorithm to RHC. The main difference is that instead of picking the best point, the algorithm allows movement in a non-optimal way, meaning pick a point worse than the current one. The probability that the algorithm will allow non-optimal movement is determined by the temperature variable. A higher temperature variable means a higher chance of moving poorly. Over time, the temperature variable is decreased or cooled until an optima or minima is found. Because of this possibility of exploration, this algorithm performs better in the event of many local optima or minima.

Genetic Algorithms are different than RHC and SA for many reasons, but namely, GA tracks a population of points rather than just a single point in time. On each iteration, a percentage of the population is selected to mate or perform a crossover of attributes, and another percentage of the population is selected to mutate in its attributes. The method of determining which attributes to perform a crossover requires domain knowledge of the problem to determine how to group attributes for this step. After this, the algorithm takes the nth percent best individuals of the population to the next iteration. The best individuals can be measured on the problem's fitness function. MIMIC works in a similar way to GA because it tracks a probability distribution to sample points for each iteration. With these points, a new threshold is created so that the top nth percentage of points will be above the threshold value. Using these points, a new probability

distribution will be created and the process will be repeated until the only values left are the optima.

In order to run these problems, I used the *ABAGAIL* problem suite to train the algorithms because it has a fully functioning suite of optimization algorithms and a full test suite with optimization problems.

Neural Net Weight Determination

This paper will be analyzing the car evaluation dataset that was analyzed in the last paper. This was done in order to provide a more thorough analysis on the data. The data came from UC Irvine and the problem was classifying the acceptability of a car given the characteristics provided within the datasets. The acceptability of the cars was ranked as unacceptable, acceptable, good, or very good. Each car also had a number of attributes associated with it, such as the purchase price, the maintenance level, the number of doors, the number of people it fits, the size of the luggage boot, and the safety estimation of the car. The purchase price and maintenance level of the car were categorized as low, medium, high, or very high. When processing the data, to make the training easier, I converted each of these values into a single value ranging from 1 to 4 respective of their categorization. The luggage boot size and safety estimation were categorized as low, medium, and high, so I converted each of these values into a single value ranging from 1 to 3 respective of their categorization. The number of doors and number of people categories used number representations except for their highest categorization. To account for this, I replaced the string to represent the highest classification with a number proportionally greater than the previous classification to make training easier. These attributes are all relevant because domain experts, at the time the data was collected, would describe the quality of a car based on these attributes. Regarding the data, there are 1728 instances of data with 6 attributes. In the last paper, I performed a hyperparameter analysis for the neural net to determine its most optimal structure, which was determined to have 35 hidden layers.

Backpropagation

The backpropagation was performed in the previous algorithm using the optimal hyperparameters generated from the 5-fold cross validation. In addition, the training versus testing error was compared in the previous paper as well.

It can be seen that in Figure 1, 35 hidden layers is the optimal number of layers to generate the lowest error rates. After 35 hidden layers, the neural net began to overfit the data in the validation phase of the cross validation. Per Figure 2, it can be seen that the optimal training percentage for the neural net was around 90% of the training samples. After training 90% of the training samples, the model began to overfit the data slightly as the error increased. When looking at the learning curves, it can be seen that neural net model achieved an error rate of around 2.5%. This is the baseline algorithm to compare against the random optimization algorithms that will be discussed for the remainder of this problem. The new algorithms will produce the weights for the neural net when training it against all of the training data, and then the testing error will be compared to the backpropagation algorithm's testing error.

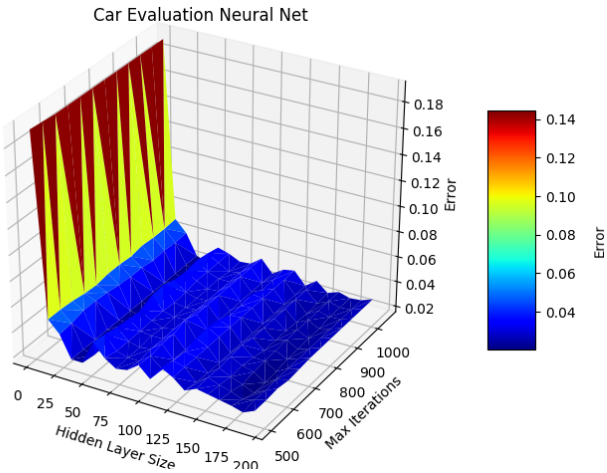

Figure 1: Neural Net Cross Validation Graph

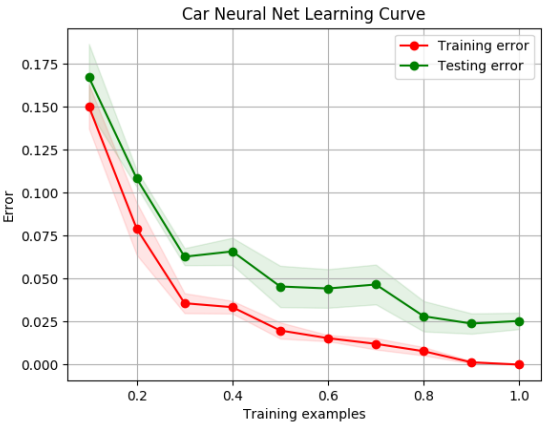

Figure 2: Neural Net Learning Curve

Random Hill Climbing

The random hill climbing algorithm works by picking a random nearby point, evaluating that point, and then attempting to progressively move to better points. Because of this, there are no real hyperparameters that need to be selected for this algorithm. To demonstrate the performance over time, the error rate of the random hill climbing algorithm was tested at varying number of iterations. The below curve represents the RHC algorithm's error rate at various iteration numbers.

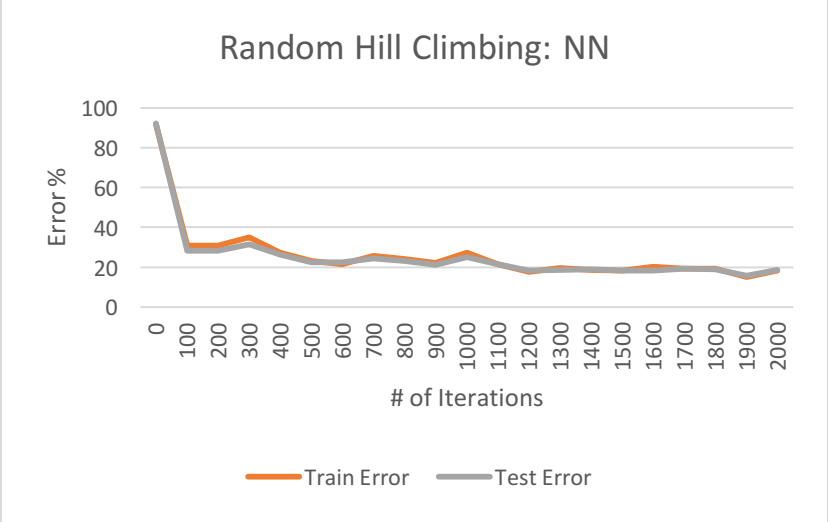

Figure 3: Neural Net Train vs. Test Error: Random Hill Climbing

Overall, this graph is fairly stable. Given that the algorithm only picks to move to points that were better than its previous point, it is understandable that the overall trend of the graph will have a negative slope. In the cases where the graph shows some positive trends, this is likely due to the fact that the final result of the RHC algorithm is solely dependent on its starting position. If the algorithm selects a bad starting point, this would result in a higher error rate, which can be seen in the graph.

When comparing RHC to backpropagation, it can be seen that RHC performed with an error rate around 13%-15% worse than backpropagation seeing how RHC had an error rate that stayed around 15%-18%. Although RHC performed worse than backpropagation, RHC performed more than 2x faster than the backpropagation algorithm. Based on the spikes in the error rate, this presents the idea that there are many local optima within the dataset, which would make it difficult for RHC to find a global optimum.

Simulated Annealing

This algorithm is similar to random hill climbing; however, it allows exploration to points that are worse than the predecessor. In order to run the SA algorithm, there were a few parameters that needed to be selected: temperature and cooling rate. In order to determine the best hyperparameter for this algorithm, I used a 5-fold cross validation algorithm to compare every combination of these parameters to pick the combination that presents the lowest error rate. The cross validation was used as a way to verify that the parameters chosen would not overfit the data. The best temperature selected was 100,000 and the best cooling rate selected was 0.9. These parameters were plugged into the algorithm and the error rate of the SA algorithm was tested at various iterations like before. The below graph demonstrates SA's error rate at different iteration numbers.

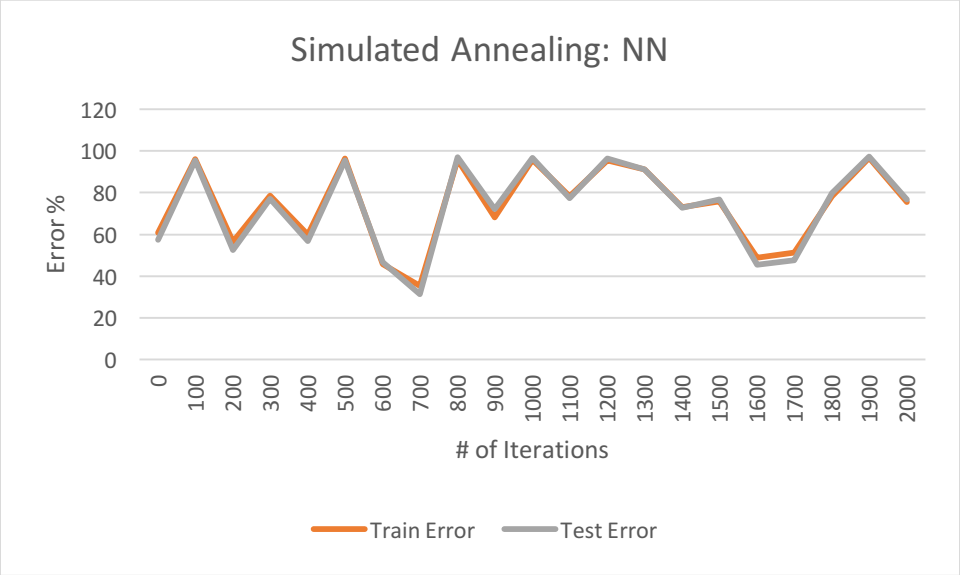

Figure 4: Neural Net Train vs. Test Error: Simulated Annealing

When looking at the graph, it is much less stable than the RHC algorithm. This is understandable because similar to RHC, it is susceptible to bad starting locations. In addition, SA allows for the algorithm to explore points that are worse than its predecessor in hopes of not getting stuck in local optima. Because of this, SA can sometimes end up in worse locations because it is willing to entertain worse points. When SA starts in a good place and has a good final result, the algorithm results in an error rate around 30%-32%. This algorithm performed much worse than the RHC algorithm, but performed at relatively the same speed as the RHC algorithm. It was faster than back propagation, but a little bit slower than RHC because with each step, it had to

evaluate whether or not it will go to a worse point. Based on the large fluctuations in the graph, this presents the idea that there are many local optima within the algorithm, which makes it difficult for the SA algorithm to find and converge at the global optimum. Given how this algorithm performed significantly worse than the RHC algorithm, it can be seen that when comparing the two algorithms, SA would not be a very good fit for this dataset to determine neural net weights.

Genetic Algorithms

This algorithm is much different than the previous two algorithms. In addition, it also had 3 hyperparameters that needed to be tuned including: the percentage of the population who mate, the percentage of the population who mutate, and the percentage of the population that move onto the next generation and iteration. In order to tune these hyperparameters, I use a 5-fold cross validation on the training set in order to reduce overfitting on the data and create the best combination of hyperparameters that produce the lowest error rates. The best hyperparameter combination was 25% population survival, a 4% mating rate, and a 4% mutation rate. Because each iteration of the genetic algorithms takes a longer time, the range of iterations was shortened. The graph below represents the error rate of the GA algorithm performed on a varying number of iterations.

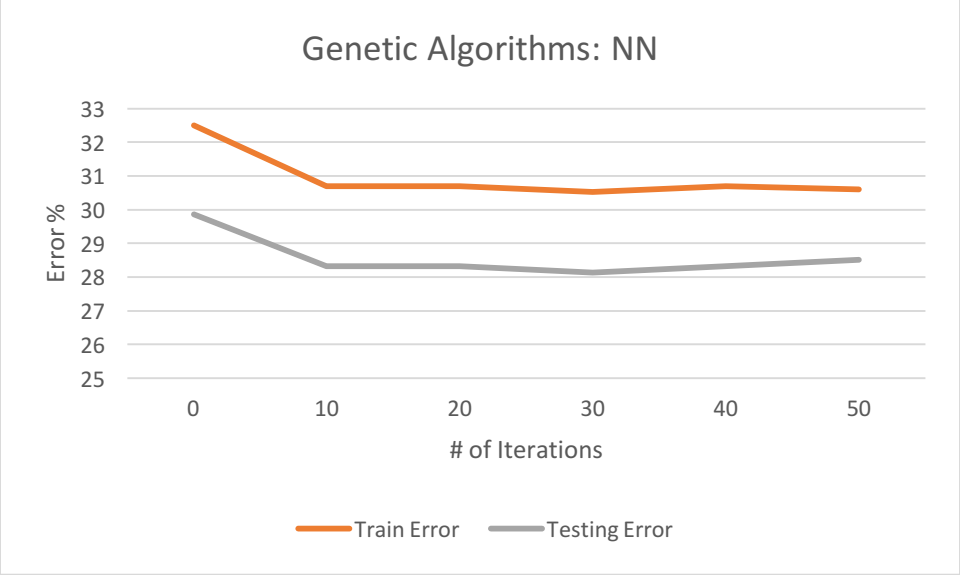

Figure 5: Neural Net Train vs. Test Error: Genetic Algorithms

When looking at this graph compared to the previous algorithms, it can be seen that this graph is much more stable (meaning it does not have many spikes error rates). This is likely a result of the actions performed within the algorithm, since it is capable to analyze and reduce the hypothesis space quickly. The mating rates and crossovers that occurred from mating resulted in limited volatility within the error rates. At its best performance, the GA performed at an error rate between 28%-29%. When comparing the error rates between the 3 algorithms, this error rate resided in the middle between them. GA has a preference bias for groups of features due to the actions of crossovers and mutations. With this preference bias in mind, it can be seen that individual attributes within this dataset might have more of an impact on the result of the dataset

than groups of the attributes because the error rate only differed slightly from the SA algorithm. This algorithm took 10x as long as the RHC algorithm to train the neural net because each iteration required more time to complete, and the algorithm had to analyze an entire population of instances rather than a single point. With this in mind, given the slow performance speed and its presented error rate, this algorithm might not hold up as the best algorithm for this dataset.

Neural Net Summary

Based on the graphs presented in this past analysis of the 3 random optimization algorithms, there is a clear algorithm that could be deemed as best. The RHC algorithm for determining the neural net weights resulted in a significantly lower error rate than the other algorithms. In addition to this, because of the algorithm's simplicity, the RHC algorithm trained the neural net in a much faster time than the other algorithms. Typically, the RHC algorithm has drawbacks when it comes to most fitness functions because of the issues it has with local optima. However, based on the end results of the RHC graph, it can be seen that this fitness function did not have many local optima that differed greatly from the global optima because it resulted in the lowest error rates outside of the other algorithms. While the GA and the SA algorithms reached reasonable error rates, RHC converged to a better error rate in a much faster time as well.

When comparing these 3 random optimization algorithms to the backpropagation algorithm, it can be seen that the backpropagation algorithm performed significantly better than the random optimization algorithms. For this dataset, it can be demonstrated that supervised learning seems to perform better because of its lower error rates; however, with more iterations, the error rates for the random optimization algorithms could have improved given how they have a negative slope. One major difference between the backpropagation learning curve and the random optimization graphs is that the random optimization graphs demonstrate a testing error that is generally lower than the training error, which means that the neural nets developed from these algorithms generalize better for the data. Overall, even though backpropagation performed better than RHC, RHC is the best random optimization algorithm for this dataset and learning structure.

Comparing Algorithms

I have chosen 3 optimization problems, Flip Flop, Knapsack, and Max K-Coloring, as a way to showcase the various strengths and weaknesses of the optimization algorithms being compared in this paper. Each of the 4 algorithms was run on the problems that will be described in their respective sections.

Flip Flop

This goal of this optimization problem is to maximize the number of alternating 1's and 0's within a bitstring. Based on this goal, the fitness function essentially evaluates the number of alternations that occur within the bitstring, so it will try to maximize the number of alternations. This is an interesting problem because of its low complexity and its large number of local optima that exist within the fitness function.

To test this optimization problem, I ran two variations of the problem. In the first problem variation, I varied the number of iterations the algorithms could use to try to converge to the optimal value. In this test, the bitstring length was kept at a constant of 800. In the second variation of the problem, I varied the bitstring length from 1 to 1000 bits. Within this problem, I kept the number of iterations constant at 20,000 due to the computational power that this number of iterations required.

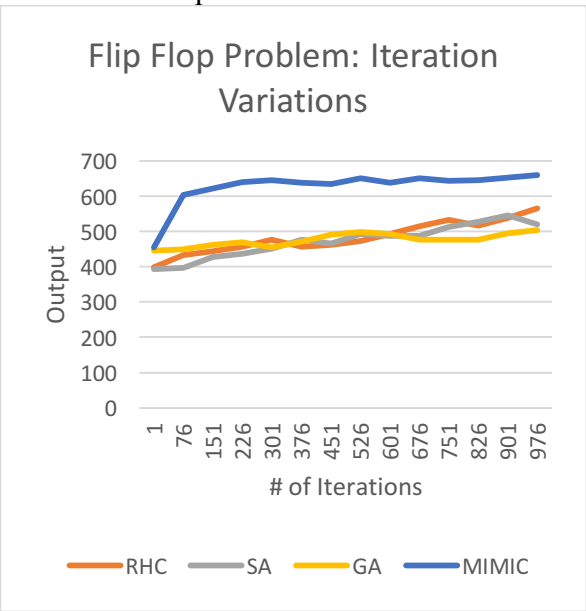

Figure 6: Flip Flop: Iteration Variations

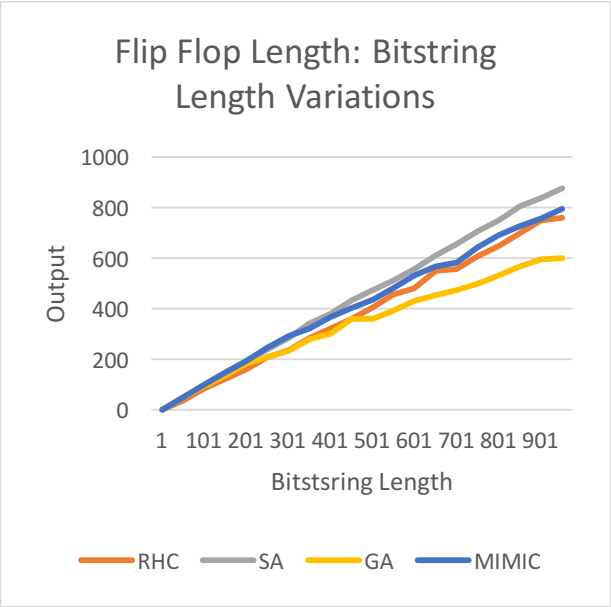

Figure 7: Flip Flop: Bitstring Length Variations

When analyzing the graphs, with a constant bitstring length, the MIMIC algorithm produced the best results out of the other algorithms; however, it produced the second best results when the bitstring length was varied. With a constant bitstring length, RHC, GA, and SA all produced similar end results that were much lower than the MIMIC algorithm. When varying the bitstring length, SA produced the best results with RHC falling right behind MIMIC and GA producing the worst results across each of the problem variations. Given this information, one would assume that MIMIC is the best algorithm of choice for this problem because of its results; however, MIMIC took almost 40-50x longer than any of the other algorithms to run. With this in mind, MIMIC might not be the best algorithm to represent this dataset. On the other hand, SA performed the best out of the algorithms on the varying bitstring lengths. Because SA was given 20,000 iterations to converge on this problem variation, this could present the idea that SA might have eventually produced better results than MIMIC if allowed more iterations to converge to an optimal value based on the positive trend shown in the first graph. Because the optimal number of alternating bits places a large interest on dependence between two bits, it would make sense that the dependency structure of MIMIC resulted in the best results for a tightly bound iteration number. However, due to MIMIC's lengthy time of convergence as a huge tradeoff, SA ultimately performs the best on this problem.

Max K-Coloring

The Max K-Coloring problem is a graph optimization problem. The idea behind this problem is to "color" or label a maximum number of vertices such that no two adjacent vertices share the

same color or label using only k colors/labels. For this optimization problem, 2 variations of the problem were run. The first variation of the problem changed the k-value or the maximum number of colors that can be used to label a graph. The number of iterations was kept constant at 20,000. The second variation of the problem changed the number of iterations the algorithm could perform, while keeping the maximum number of colors, K, constant.

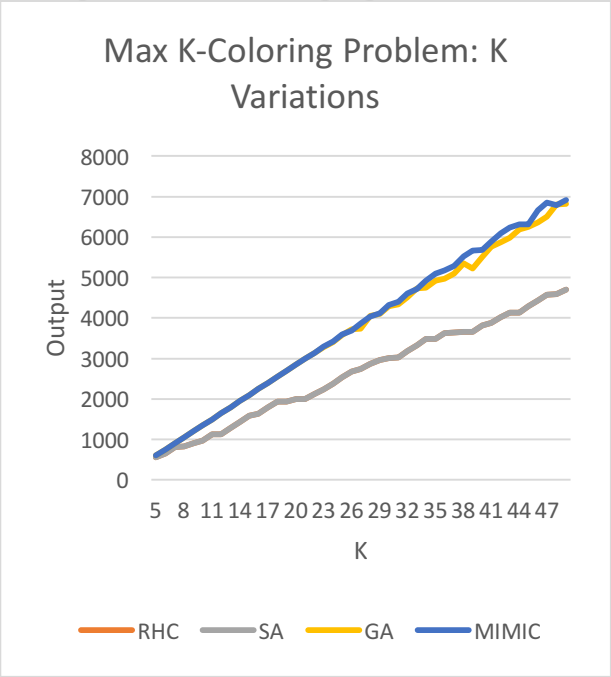

Figure 8: Max K-Coloring: K Variations

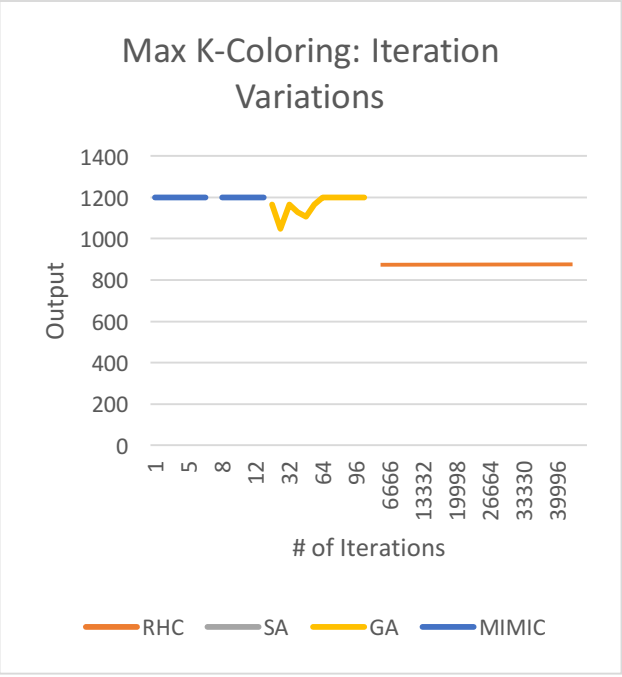

Figure 9: Max K-Coloring: Iteration Variations

When looking at the above graphs, the clear optimal algorithms are either MIMIC or Genetic Algorithms for this problem. RHC and SA had a much worse performance than MIMIC and GA; however, they actually produced the exact same output across both problem variations. MIMIC and GA produced the best outputs. In the second problem variation, the number of iterations for Genetic Algorithms and MIMIC were limited due to computational constraints; however, this iteration restriction did not have a significant impact on the overall results of the algorithm because they still outperformed RHC and SA. MIMIC was able to perform so well on this problem because it was able to exploit the dependency relationships between the graph vertices. GA was able to perform well for similar reasons, as it was able to use crossover as a way to deal with the connected nature of the graphs. When the number iterations were used as the independent variable, each of the algorithms produced their optimal result in around the same time. This was because the number of iterations for MIMIC and GA were limited to ensure the time equality. Despite these restrictions, MIMIC and GA still performed the best. When the K-value was the independent variable, MIMIC took the longest twice as long as the other algorithms to converge to an optimal result. Overall, based on their relatively equal performance, and the time tradeoffs, GA is the best random optimization algorithm for this problem.

Knapsack

The knapsack problem is a very common problem in the computer science community. It is a problem based on combinatorial optimization. Given a set of items that each have a specific

weight and value associated with them, determine the amount of each item that one should include in the "knapsack" that stays within the maximum allowed weight and provides the most value as well. This is an interesting problem because of its increased complexity, due to the increased number of variables, and because of its large number of local optima within the optimization function.

For this problem, 2 variations of the problem were run. The first variation of the problem was changing the maximum weight allowed within the "knapsack". In this problem, the various random optimization algorithms were allowed to run for 20,000 iterations at each of the maximum weight values allowed. In the second variation of the problem, the optimization algorithms were run at a different number of iterations for a standard maximum weight of 3200.

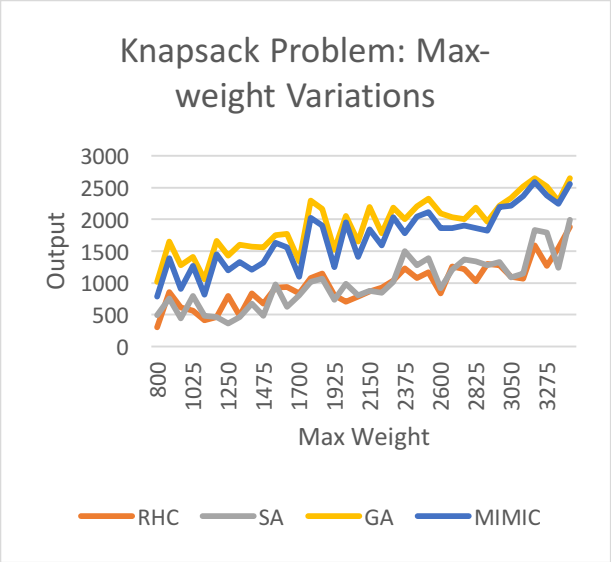

Figure 10: Knapsack: Max-weight Variations

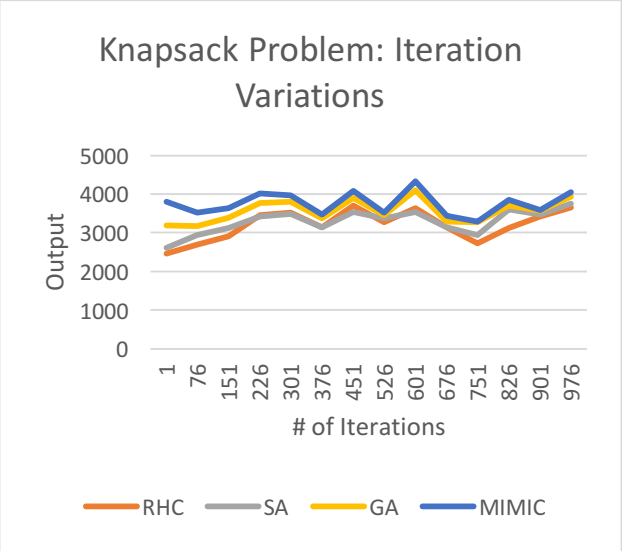

Figure 11: Knapsack: Iteration Variations

When looking at the graphs above, MIMIC and Genetic Algorithms produced the best results overall. When the maximum weight of the knapsack problem was changed, Genetic Algorithms produced slightly better results, whereas in the constant maximum weight, MIMIC produced slightly better results. This is likely due to the fact that MIMIC has the ability to filter out more points with each iteration than GA, so this could be the reason that GA did not perform as well when its number of iterations were limited. However, like the previous problems, RHC and SA both performed in the shortest amount of time resulting in only slightly different end results and finishing times. Based on the graphs and the large spikes that exist within the RHC and SA lines, this proves the idea that there were many local optima within the knapsack optimization function. GA took 5x as long as RHC and SA to converge to one of the optima, whereas MIMIC took around 25x as long as the other algorithms to complete. It was expected that GA and MIMIC would perform better than RHC and SA because GA has the ability to exploit the relationships between objects to help find a local optimum, and MIMIC also performed well because of its ability to exploit dependency relationships as a way to filter down to the optima faster. Overall, GA is the best choice for this optimization problem because it performs with similar results to the MIMIC algorithm, but it finishes in 1/5 of the time of MIMIC, so the faster one is preferable.

Conclusion

The car evaluation dataset, the Flip-Flop problem, the max K-coloring problem, and the knapsack problem all provided different ways of comparing the different random optimization algorithms. With regards to the neural network problem, it allowed a demonstration of a direct comparison between how random optimization works with regards to selecting neural net weights as well as how it compares to gradient descent. When looking at the problems as a whole, the nature and structure of the problems led to better performance for certain algorithms, which ultimately highlights some of the strengths and weaknesses with each of them. Based on the design of each of these algorithms, they each have problems that can best suit them. This provides insight into the idea that knowing more details about the problem can lead to a better randomized optimization algorithm selection, which can result in less computational power requirements and better performance for the problem.

References

Code: https://github.com/tobincolby/CS4641-RandomOptimization
Algorithms: *ABAGAIL + Documentation* – https://github.com/pushkar/ABAGAIL
NN Testing Suite: https://gist.github.com/mosdragon/53edf8e69fde531db69e
Car Evaluation Data: https://archive.ics.uci.edu/ml/datasets/Car+Evaluation