# Randomized Optimization

Rohan Ramakrishnan
Georgia Institute of Technology
rohanrk@gatech.edu

***Abstract*—This analysis comprises two sections. In the first section, neural networks are trained using three different randomized optimization algorithms and tested on a breast cancer dataset and performance is analyzed between all three. In the second section, four different randomized optimization algorithms are used to solve three different optimization problems: N-Queens, Four Peaks, and Knapsack. Each problem highlights the strengths of one of the optimization problems. Their performances are measured and compared to show the types of problems best suited for each algorithm.**

## I. Introduction

In practice, neural networks utilize the backpropagation algorithm along with gradient descent to optimize their weights and converge to the best fit weights. This analysis aims to answer the question: How does performance change if randomized optimization algorithms (ROAs) are applied to train neural networks instead of traditional backpropagation? For the purposes of this analysis, a breast cancer dataset is used to train and test neural nets trained on different ROAs. This breast cancer dataset is split into 200 training instances and 89 testing instances with each instance contains nine attributes. Each neural network is trained using six hidden layers while the number of training iterations are varied. The implementations in the open source library ABAGAIL were used to compile all randomized optimization results.

## II. Neural Networks

### A. MultilayerPerceptron Network

For comparison, figure 1 shows the performance of a neural net trained using backpropagation when applied to the breast-cancer dataset. These results were generated using Weka's MultilayerPerceptron classifier and provide a good basis of comparison for the networks trained using ROAs. Through trial and inspection, a learning rate of 0.1 and momentum factor of 0.4 were deemed to be the optimal hyperparameters for the breast cancer dataset. Each ROA was run six times on the same dataset with varying the number of training iterations and 6 hidden units. It is important to note that the learning rate is encapsulated within the library so the learning rate that ABAGAIL uses is unknown. Training and testing error measurements were calculated using sum of squared error for each of 5000 training iterations. Another important comparison is the difference in network units. Weka uses traditional sigmoid units whereas ABAGAIL uses hyperbolic sigmoid units. These are units defined by Equation 1.

$$f(x) = \tanh(x)$$

*Equation 1: Hyperbolic Tangent Activation Function*

These units rescale the sigmoid unit's range so it outputs values between -1 and 1 instead of from 0 to 1.

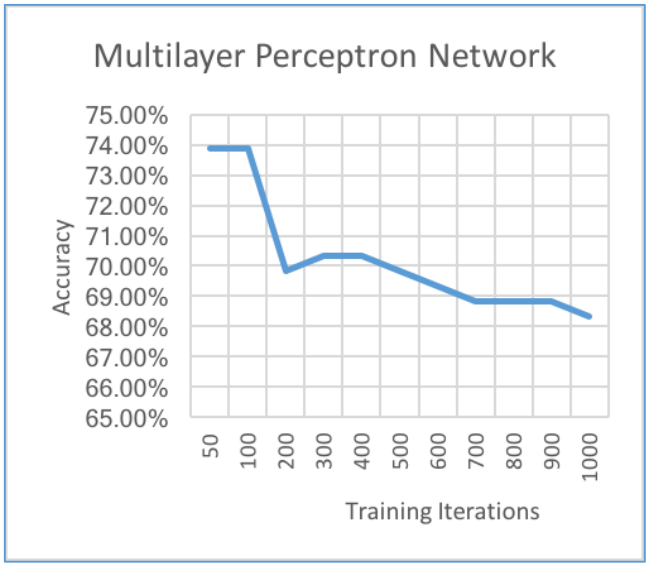



*Figure 1: Accuracy vs Training Iterations of Multilayer Perceptron Network on Breast Cancer Dataset*

### B. Randomized Hill Climbing

Hill Climbing is a simple optimization strategy that finds a solution to a give optimization problem and iteratively improves this solution by comparing its solution with the solution of the best neighbor. Randomized Hill Climbing (RHC), also known as Stochastic Hill Climbing, applies a similar strategy, opting to select a random neighbor instead of the best neighbor. For the purposes of this analysis, RHC does not implement random restarts to highlight the sharp contrasts in performance between trials when the algorithm gets stuck in local minima.

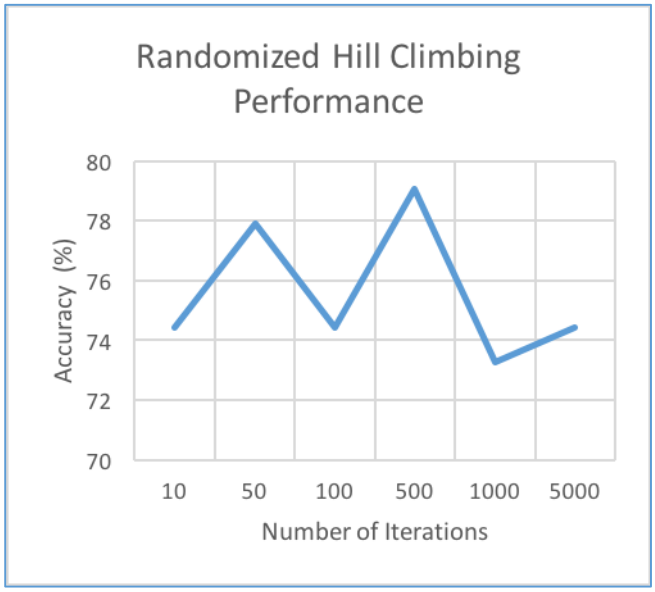



*Figure 2: Performance of Neural Network on Breast Cancer Dataset*

Figure 2 shows some interesting results. The first is that RHC immediately outperforms the MultilayerPerceptron classifier. Weka's waning accuracy as the training iterations increase is a sign of the neural net overfitting to the training data. RHC achieves higher accuracy in significantly fewer training iteartions as well. At 10 iterations, RHC is on par with Weka's classifier. However, RHC is easily susceptible to local optima and this can be observed by the sharp increases and decreases in accuracy between trials. This trend implies that RHC is trapped in local minima during several trials and is stuck at those points. This does cause the neural net to overfit, though it still achieves better results than Weka's classifier whose performance incrementally decreases as the number of iterations increase. The training and testing error showcase through random spikes in error as the network trains as can be seen in figure 3.

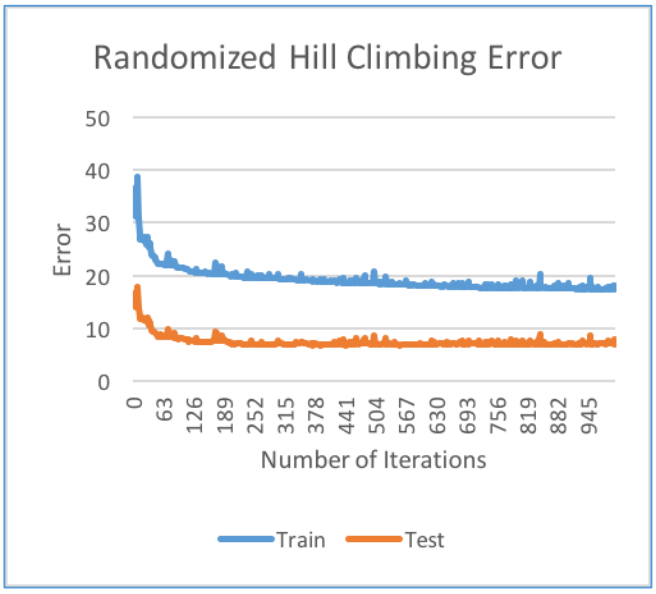



*Figure 3: Training and Testing Error of RHC trained Neural Network*

The training and testing error show that the neural net is training itself properly. The network's goal is to minimize its squared error as much as possible. Around 300 to 400 iterations, the error converges which shows that the algorithm is stuck in local optima beginning at that point and cannot optimize its error any further from that point onward.

### *C. Simulated Annealing*

Simulated Annealing is a probabilistic optimization method that is similar to Randomized Hill Climbing. However, unlike RHC, Simulated Annealing allows itself to step into non-optimal instances with the goal of circumventing local optima and finding the global optimum. For this experiment, Simulated Annealing was used with an initial temperature of 1,000,000 and a cooling factor of 0.5. Simulated Annealing performs the worst with fewer training iterations. This is consistent with Simulated Annealing's properties as the algorithm takes the first iterations to explore many non-optimal points. It is the only algorithm that does not prune less fit points as even Randomized Hill Climbing does not explore suboptimal points. The accuracy peaks at 75.581% after 100 iterations. Although the probability greatly decreases as the algorithm runs through more training iterations and the temperature decreases, there is still a nonzero probability that simulated annealing explores suboptimal points. Although the probability is low, with more training iterations, the algorithm is more likely to explore more. It is important to note that these results will vary through different trials due to the stochastic nature of the algorithm. A possible modification to this algorithm is to run the Simulated Annealing algorithm 5 times per training iteration and take the average of the optimal weights found in all those trials. This would give more consistent results and a better idea of how effective Simulated Annealing works on this dataset.

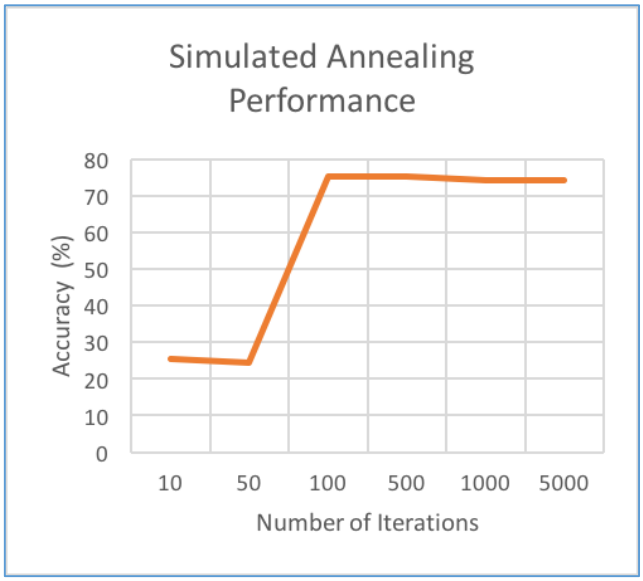



*Figure 4: Performance of Neural Network trained using Simulated Annealing on Breast Cancer Dataset*

Figure 5 shows the training and testing errors over 1000 iterations. In this run, the training and test error

measurements start lower than the other algorithms but get worse during the first few iterations. This is consistent with Simulated Annealing's initial exploration strategy. The error quickly drops as the neural net begins to learn and exploit its knowledge rather than explore suboptimal weights. This can be compared to reducing the learning rate though it has different effects and reducing the learning rate enforces a constant step on how quickly a neural net changes when it encounters errors. There are also several spikes that represent the training iterations where simulated annealing chooses to explore random suboptimal points to find the global optimum. There are significantly more spikes at the beginning but the curve becomes smoother as the temperature decreases and the neural net exploits its knowledge and keeps its current optimal weights. Similar to the RHC error curves, these error curves also slowly converge since the neural net cannot optimize its weights any further.

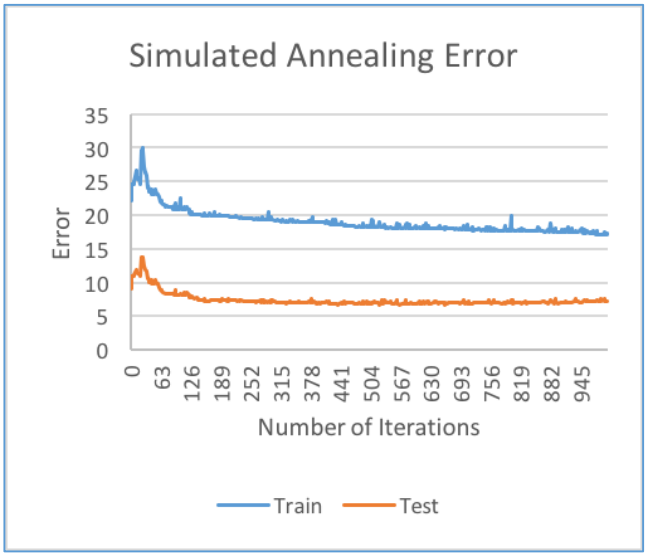



*Figure 5: Training and Testing Error of Neural Net trained using Simulated Annealing on Breast Cancer Dataset*

### *D. Genetic Algorithms*

Genetic Algorithms (GAs) are a class of optimization algorithms motivated by the biological concepts of natural selection and genetic recombination. The standard GA randomly selects candidate solutions from an initial population. Then, the GA encodes candidate solutions as fixed length vectors and allows them to evolve through a number of generations. At each generation, the GA measures the fitness of each candidate solution, selects the best or most fit candidates and uses recombination and mutation to generate a new population. Based on their fitness, the algorithm selects certain data points. For the purposes of this paper, only the standard genetic algorithm is used. The neural network was trained with an initial population size of 200 with 25 mates per iteration and 100 mutations per iteration.

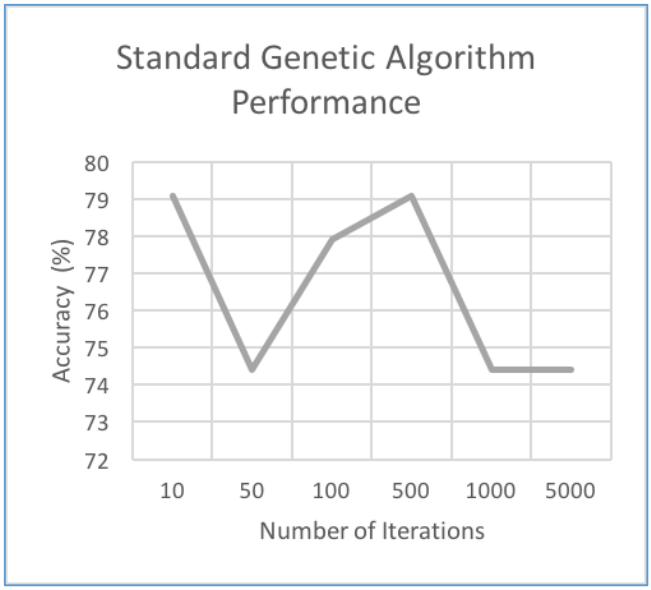



*Figure 6: Performance of Neural Net on Breast Cancer Dataset when trained using Standard Genetic Algorithm*

The breast cancer has only nine attributes so there are not much room to separate and optimize each attribute. Furthermore, some of the attributes themselves are not independent. An example of this is degree of malignancy and tumor size. If the genetic algorithm attempted to optimize based on the values of these attributes separately, it could make errors during crossover and mutation.

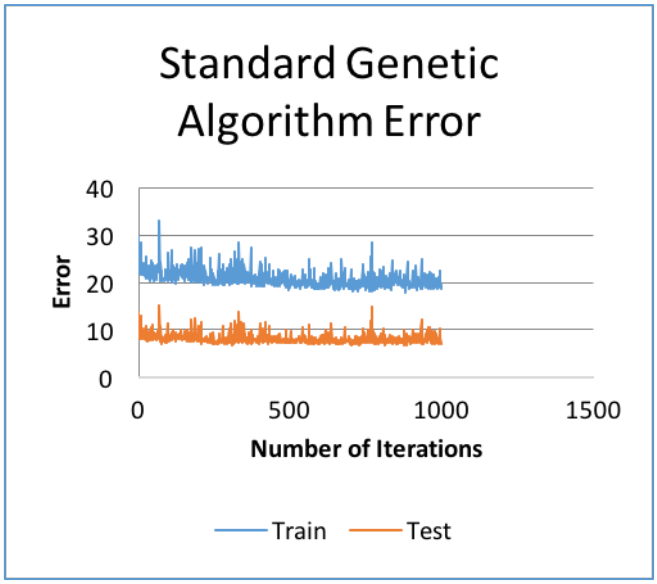



*Figure 7: Training and Testing Error of a Neural Network trained using a Standard Genetic Algorithm on Breast Cancer Dataset*

The error curves in figure 7 also reveal a very high amount of variance between the error as every generation passes. The logical conclusion here is that the Standard Genetic Algorithm performs poorly on this dataset despite comparable accuracy with the other algorithms. This high accuracy is due to the size of the dataset. If this had been run on a larger dataset the difference in performance would be far more evident. A possible modification to this algorithm

would be to group dependent attributes together and optimize them at the same time. This allows the genetic algorithm to highlight its strength by optimizing independent attributes separately while also optimizing dependent attributes together and combining the results of all the attributes of an instance to step to the global optimum. This discussion has mainly focused on accuracy and error. Figure 8 shows another relevant metric which is the amount of time the neural network takes to train. As expected, The Genetic Algorithm takes the longest training time since it does more work per iteration. For this dataset, this is particularly bad as the Genetic Algorithm takes around 60 seconds to train a neural net that has similar performance results as those trained in 4 seconds using the other algorithms.


*Figure 8: Training Time of Neural Nets using ROAs*

### *E. Concluding Remarks*

Ultimately, we can see that the performance of neural networks trained using randomized optimization algorithms trump the performance of those trained with standard backpropagation for this dataset. Although ABAGAIL's neural net driver is tuned for binary datasets, it would be interesting to compare results on a multiclass dataset such as optdigits. Weka's classifier performed very well on this dataset, so it would be an interesting comparison to see how the ROAs perform and if they continue outperforming Weka's MultilayerPerceptron Network or if they run into roadblocks attempting to optimize the neural network's weights. Naturally the hyperparameters would have to be retuned due to the No Free Lunch theorem. It is also important to note that the size and features of this dataset plays a large role in performance measurements. Firstly, since the dataset is so small, it's possible that the accuracy values are somewhat inflated as the test set only introduces 89 unseen instances into play. Furthermore, the dataset is significantly unbalanced which also factors into these results. Nevertheless, one can observe an example of a supervised learning problems where utilizing random optimization to tune a classifier yields better results than more traditional methods such as gradient descent and the delta rule.

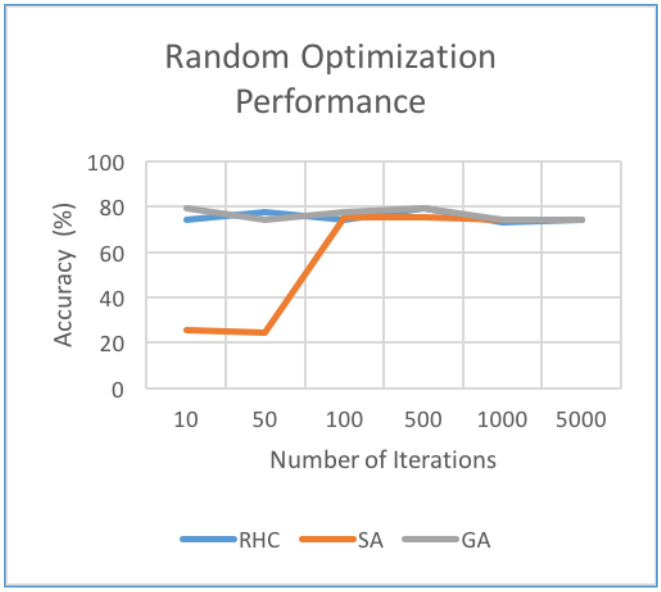



*Figure 9: Performance Comparison of all ROAs*

## III. Optimization Problems

### *A. Overview*

Each optimization problem in this report have varying strengths and weaknesses. Although ML is plagued by the effects of the No Free Lunch theorem, an important skill for any machine learning student is the ability to decide and choose the best algorithm to solve any given problem. This section observes the performance of four optimization algorithms on three different problems to showcase which algorithm performs the best and describes the optimal types of problems for each ROA.

### *B. Traveling Salesman*

The Traveling Salesman problem is a classic problem in computer science. Given a list of cities and the distance between pairs of cities, find the path that visits every city and minimizes the distance. This problem cannot even be checked in polynomial time which is evidence to how costly the evaluation function is. ABAGAIL's evaluation function is measured as the reciprocal of the minimal solution. Since it is a reciprocal, an algorithm that finds a minimized distance will maximize this fitness function. This problem is interesting as it provides the opportunity to view the performance of ROAs on a very difficult and relevant problem with several path planning applications. In the results, The Standard Genetic Algorithm outshines the rest in performance. This is because the Genetic Algorithm can optimize elements of each partial path, combine them together and converge to an optimal path. Another factor to note is ABAGAIL's encoding. ABAGAIL uses a sort encoding for MIMIC as opposed to the permutation based encoding for the other ROAs. This adds an extra O(N) step to the algorithm (though this is not the dominating factor, it is worth noting as it factors into the overall runtime). This encoding does not affect Figure ... shows the results of all algorithms on the problem. The number of cities is varied through all tests. Randomized Hill Climbing and Simulated

Annealing were run for 200,000 iterations. Initial temperature and cooling rate were 1E12 and 0.95 respectively. Standard Genetic Algorithm and MIMIC were run for 1000 iterations. For Standard Genetic Algorithm, the initial size was set to 200, mates per iterations was set t0 150, and mutations per generation was set to 20. MIMIC was set to generate 200 samples and keep the 100 most fit samples per iteration.

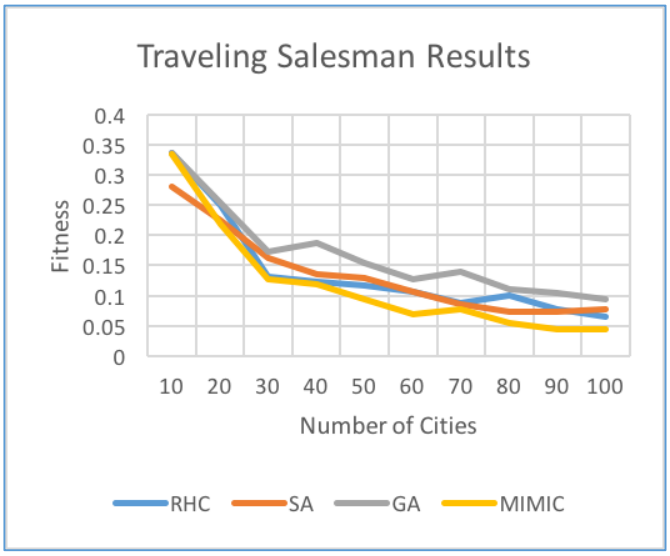



*Figure 10: Fitness Results of all ROAs applied to the Traveling Salesman Problem*

The Standard Genetic Algorithm also runs very quickly on this problem. The runtime increase is insignificant which is very impressive considering the nature of this problem and how much more work is done per iteration for Genetic Algorithms. This runtime would increase further if the number of iterations were increased. It is important to note that the Genetic Algorithm was run for 1000 iterations compared to the 200,000 iterations Randomized Hill Climbing and Simulated Annealing went through. Figures 11 and 12 show the runtime of the ROAs when applied to this problem. MIMIC's runtime is drastically large which highlights the heavy cost of running the evaluation function and iteratively generating samples.

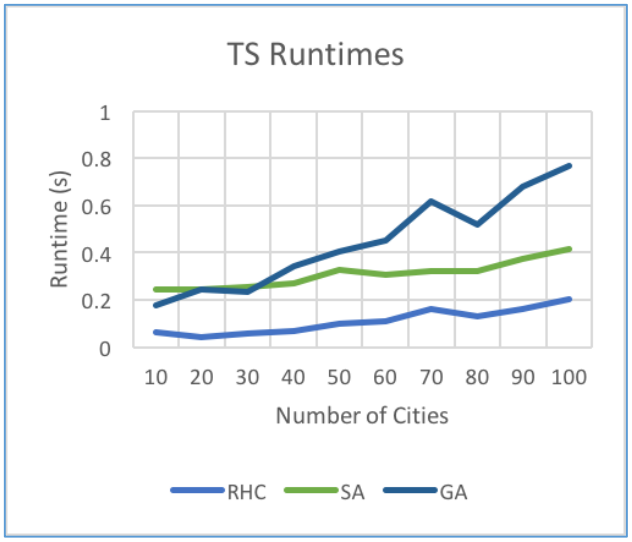



*Figure 11: Traveling Salesman Runtimes for RHC, SA, and GA*

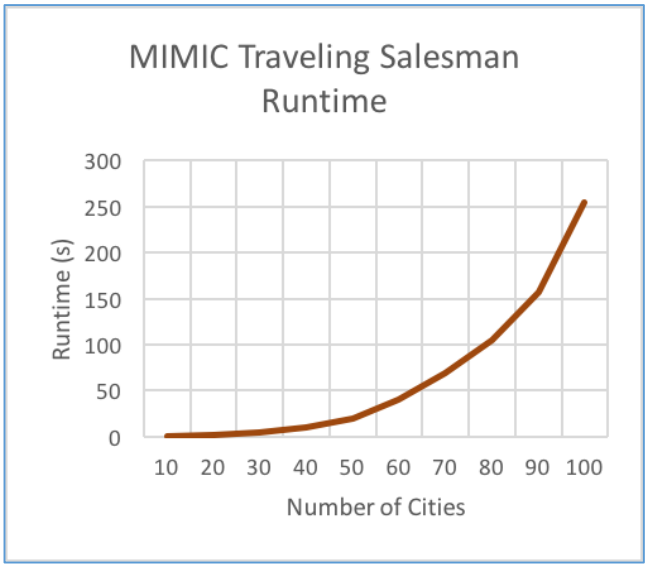
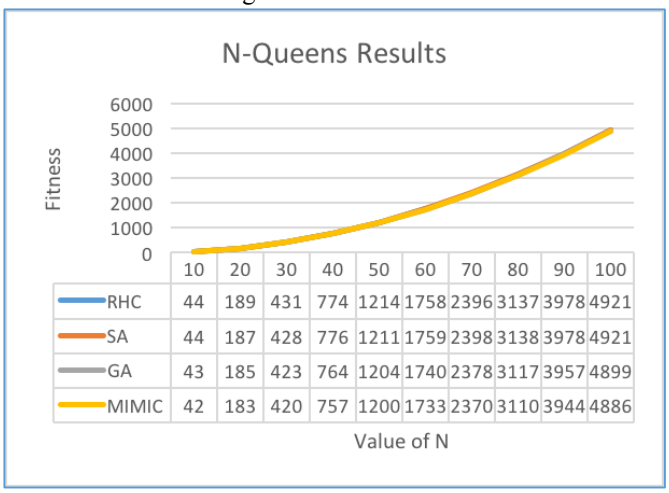



*Figure 12: MIMIC's runtime when applied to the Traveling Salesman Problem*

### *C. N-Queens*

The N-Queens problem is defined as follows. Given an NxN chessboard, place N queens in such a way that no queen can capture another. It is important to note that this optimization problem is not plagued by local optima since a configuration of queens is either legal or not legal. The fitness function used to evaluate the effectiveness of any solution is defined as the number of non-attacking queens on the board. This is a cheap function to calculate. ABAGAIL's implementation evaluates any given solution in $O(N^2)$ time by checking each pair and ensuring that they are not attacking each other. Another observation about the fitness function is that it contains no local optima since it is a monotonically increasing function in terms of N. Due to these properties, the N-Queens problem presents an interesting challenge that highlights the strengths of Simulated Annealing. Simulated Annealing works well on problems with fewer local optima since it decreases the likelihood that the algorithm will trap itself in local optima. N-Queens ensures that Simulated

Annealing will never get trapped. This guarantee implies that Simulated Annealing is the best algorithm to solve the N-Queens problem as it runs faster and generates solutions faster than other algorithms. Each algorithm was trained with 100 training iterations with the exception of MIMIC which was trained with 5 training iterations. Simulated Annealing was run with an initial temperature of 10 and cooling rate of 0.1. It is important to note that although the starting temperature is much lower than in the previous section, the lower cooling rate allows the algorithm to decrease its temperature more slowly which allows it more exploration time. A standard genetic algorithm is run with an initial population of 200, 0 mates per iteration, and 10 mutations per iteration. MIMIC generates 200 samples and keeps the 10 most fit samples for this problem. Figures 13 and 14 show the fitness values and runtimes with varying values of N for each algorithm.




| | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
|---|---|---|---|---|---|---|---|---|---|---|
| RHC | 44 | 189 | 431 | 774 | 1214 | 1758 | 2396 | 3137 | 3978 | 4921 |
| SA | 44 | 187 | 428 | 776 | 1211 | 1759 | 2398 | 3138 | 3978 | 4921 |
| GA | 43 | 185 | 423 | 764 | 1204 | 1740 | 2378 | 3117 | 3957 | 4899 |
| MIMIC | 42 | 183 | 420 | 757 | 1200 | 1733 | 2370 | 3110 | 3944 | 4886 |

*Figure 13: Performance of all ROAs applied to the N-Queens Problem.*

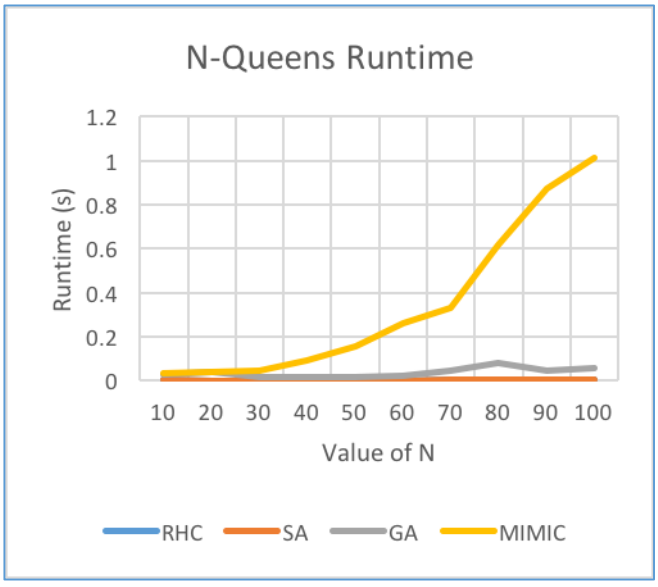



*Figure 14: N-Queens Runtimes for all ROAs*

### D. *Knapsack*

Knapsack is another classic computer science problem. Given three inputs: list of item values, a list of corresponding item weights, and a weight threshold; find the number of each item that can be put in a collection such that the weight of the items is less than the threshold and the total value of the items is as large as possible. We can express our optimization function mathematically as follows.

$$F(x) = \sum_{k=1}^{N} r_k x_k \;\; s.t \sum_{k=1}^{N} w_k x_k \;\leq W_{max}$$

*Equation 2: Knapsack Optimization Function*

Note that since the domain of F(x) is constrained to integers, maximizing F(x) is an integer programming problem which is an NP-complete optimization problem. This implies that the cost to evaluate the function for different inputs is very high. Due to this property, MIMIC distinguishes itself as the best algorithm to solve this problem. MIMIC prefers problems with a high dependence on structure and with evaluation functions with high time costs. Furthermore, MIMIC prefers problems that can represent all points between the starting point, which is a uniform distribution, and the optimal value. Knapsack gives a finite array of item values and item weights, so there is a finite combination of items that satisfy the weight constraint. Since there is a finite number of combinations of items and all of them can be represented and evaluated, MIMIC can guarantee that it avoids local optima. Even though MIMIC takes significantly longer to solve the problem, it is able to take advantage of the structure of Knapsack and find the most optimal solution.

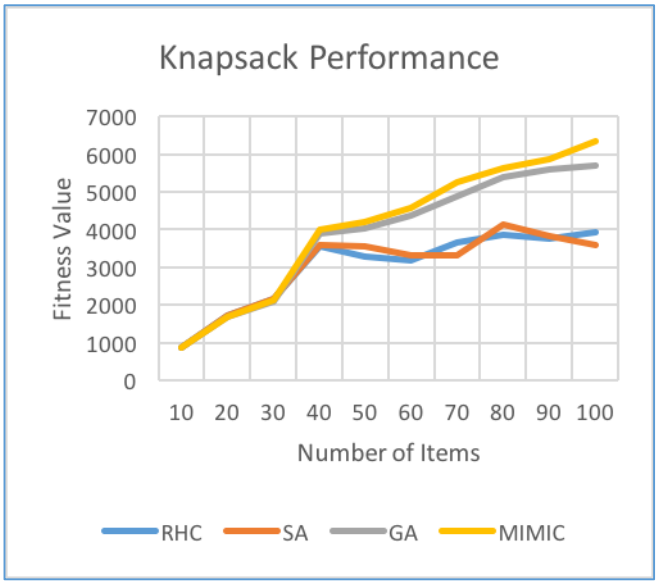



*Figure 15: Performance of all ROAs applied to the Knapsack Problem*

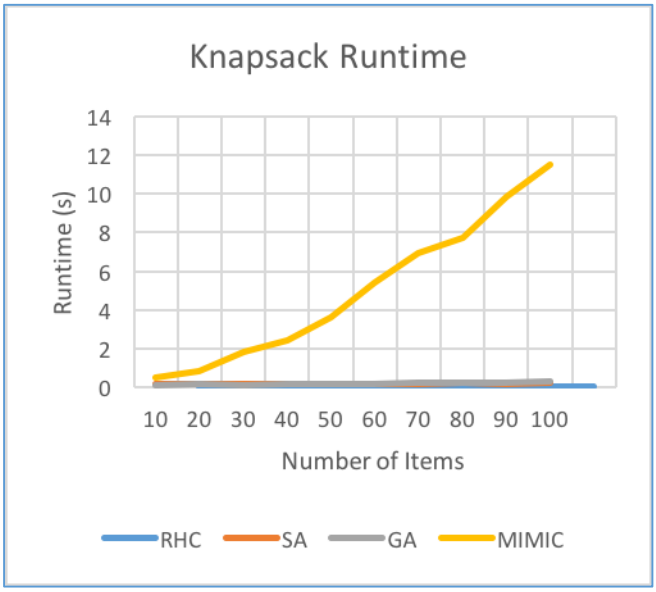



*Figure 16: Knapsack Runtime for all ROAs*

## IV. Conclusion

The goal of randomized optimization algorithms is to introduce randomness and sampling in an attempt to reduce computation time for very difficult learning and optimization problems. Through this analysis, we have achieved a more in depth look at several ROAs. We have applied randomized optimization algorithms to a familiar classification problem and proved that such methods improve performance and runtime. We have also applied these techniques to well-known computer science problems and observed the improvement in runtime while also analyzing the properties that highlight the effectiveness of different ROAs.

## Acknowledgment

Special thanks to Pushkar Kolhe and all contributors to the ABAGAIL Machine Learning library for providing several implementations and example code for running optimization algorithm tests.

## References

[1] Matjaz Zwitter & Milan Soklic (1988). UCI Machine Learning Repository [https://archive.ics.uci.edu/ml/datasets/breast+cancer]. Ljubljana, Yugoslavia :Institute of Oncology University Medical Center.

[2] Mitchell, Tom M. *Machine Learning*. McGraw Hill, 2017.

[3] Isbell, Charles L. Jr (1996) Randomized Local Search as Successive Estimation of Probability Densities. Retrieved from https://www.cc.gatech.edu/~isbell/tutorials/mimic-tutorial.pdf, February 28, 2019

[4] Baluja, Shumeet & Caruana, Rich (1995) Removing the Genetics from the Standard Genetic Algorithm. Retrieved from https://www.ri.cmu.edu/pub_files/pub1/baluja_shumeet_1995_1/baluja_shumeet_1995_1.pdf, February 28, 2019

[5] Sarkar, Uddalok & Nag Sayan (2017) An Adaptive Genetic Algorithm for Solving N-Queens Problem. Retrieved from https://arxiv.org/pdf/1802.02006.pdf, February 28, 2019

[6] Russel, Steve & Norvig, Peter. *Artificial Intelligence: A Modern Approach*. Pearson, 2010.

[7] Niu, Shu-Chen (2003). The Knapsack Problem. Retrieved from https://www.utdallas.edu/~scniu/OPRE-6201/documents/DP3-Knapsack.pdf, February 29, 2019