

Randomized Optimization

Abstract – The goal of this paper is to provide a detailed analysis of four randomized optimization algorithms: simulated annealing, randomized hill climbing, genetic algorithms, and MIMIC. The analysis of the four optimization algorithms will be done in two parts. First, there will be three interesting problems that will be analyzed: N-Queens Problem, the Knapsack Problem, and the Four Peaks Problem using all four optimization algorithms and then, we will be analyzing the results and runtime of all four optimization algorithms to determine the best algorithm for specific types of problems. Second, we will be optimizing the neural network created from the Supervised Learning assignment, but instead of using gradient descent with backpropagation for optimization, we will be using the four optimization algorithms to search for the optimal weights and analyze the results.

Three Interesting Problems

1.1 Introduction

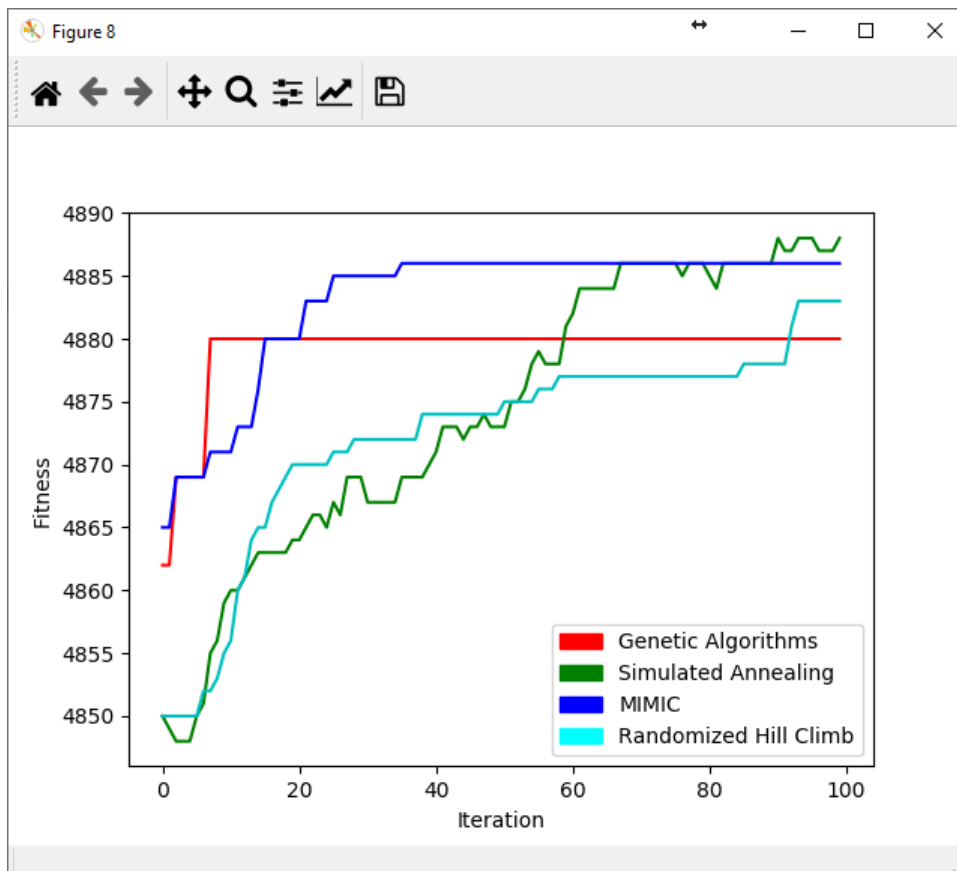
To begin the analysis of the four optimization algorithms, we will start with the famous N-Queens problem. Then, we will proceed with the Knapsack problem. And finally, we will end with the Four Peaks problem. These are all simple problems that can be easily solved for small values. But when the search space increases, and we expand to the input size to a larger and larger amount, certain optimization algorithms will start to fail start analysis will show that there is a difference in the results provided by each algorithm. For all three problems, we will be using the Python and the mlrose library for coding up the examples and analysis of the results.

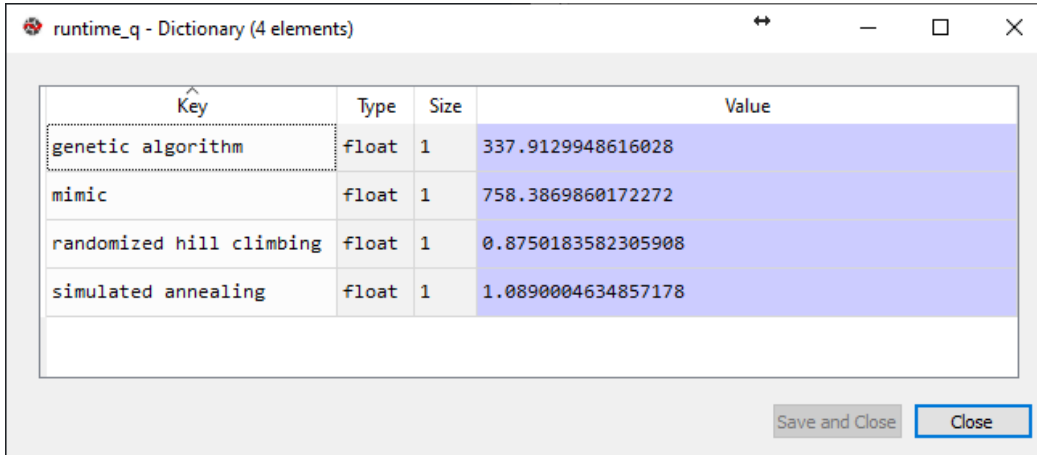
1.2 N-Queens Problem (Simulated Annealing)

The N-Queens problem is a famous NP-complete optimization problem that aims to find possible positions for n queens, where each queen will not be able to attack any other queen on the board. The rules are as follows: a queen can move any number of spaces up, down, left, right, and diagonal, and the goal is to place all n queens such that no queen is able to attack another. Analysis was done for 8-Queens, 50-Queens, and 100-Queens. For 8-Queens, the result was obtained easily for all four algorithms, but as the size of N increased, a noticeable difference between the four algorithms started to emerge. For the n-queens problem, the following parameters were used for each optimization algorithm:

Randomized Optimization Algorithm	Optimal Parameters
Genetic Algorithm	pop_size = 500, mutation_prob = 0.001, max_attempts = 500, max_iters = 100, N = 100
Simulated Annealing	schedule = GeomDecay(), max_attempts = 500, max_iters = 100, N = 100
MIMIC	pop_size = 500, keep_pct = 0.8, max_attempts = 500, max_iters = 100, N = 100
Randomized Hill Climbing	max_attempts = 500, max_iters = 100, N = 100

With the parameters specified above, my experiment yielded these results for 100-Queens:





Key	Type	Size	Value
genetic algorithm	float	1	337.9129948616028
mimic	float	1	758.3869860172272
randomized hill climbing	float	1	0.8750183582305908
simulated annealing	float	1	1.0890004634857178

From the graph above for Fitness score vs. Iteration, it is evident that Simulated Annealing performs slightly better than MIMIC but a lot better than Randomized Hill

Climbing and Genetic Algorithms. For the 100-Queens problem, examining only Fitness vs. Iterations for all four optimization algorithms provides us with only a small picture of how well each algorithm performs; by examining the runtime and combining the runtime complexity with the results provided, we get the big picture. It's obvious once we examine both the results and the runtime that Simulated Annealing is the better algorithm for the 100-Queens problem because of the fact that Simulated Annealing takes 1.09 seconds vs. Genetic Algorithms 337.91 seconds, MIMIC's 758.39 seconds, and Randomized Hill Climbing's 0.86 seconds. Why is it the case that Simulated Annealing performs better here than the other algorithms? It is because simulated annealing and randomized hill climbing are both algorithms that are faster to run per iteration, but requires more iterations to converge vs. genetic algorithms and mimic which requires more time per iteration but requires less iterations. We can tell from the graph that the problem itself isn't complicated and is rather straightforward which explains why all four algorithms reach high scores at about the same number of iterations. However, we can see that randomized hill climbing may have gotten stuck at a local optimum whereas simulated annealing found a local optimum but immediately jumped to a better result and was not stuck for as long as randomized hill climbing was. For such reasons, simulated annealing is the preferred algorithm for N-Queens because the problem is simple and simulated annealing works great for finding solutions for simple problems without getting stuck at local optimums or minimums.

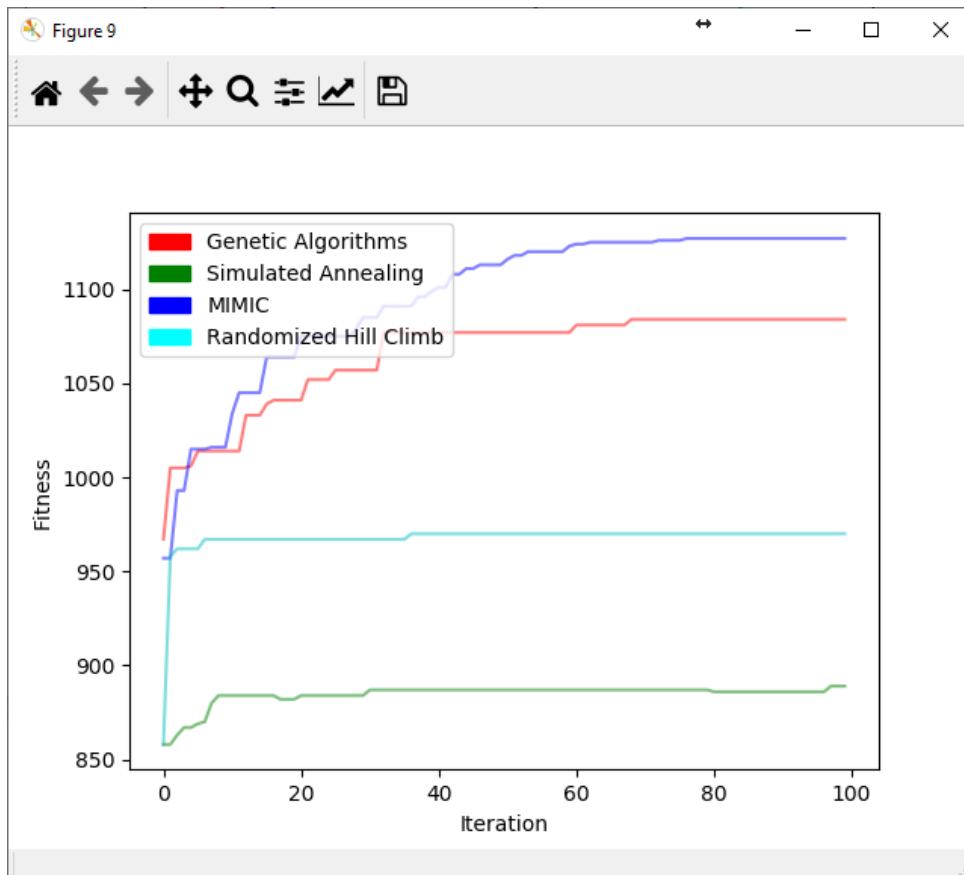
1.3 Knapsack Problem (MIMIC)

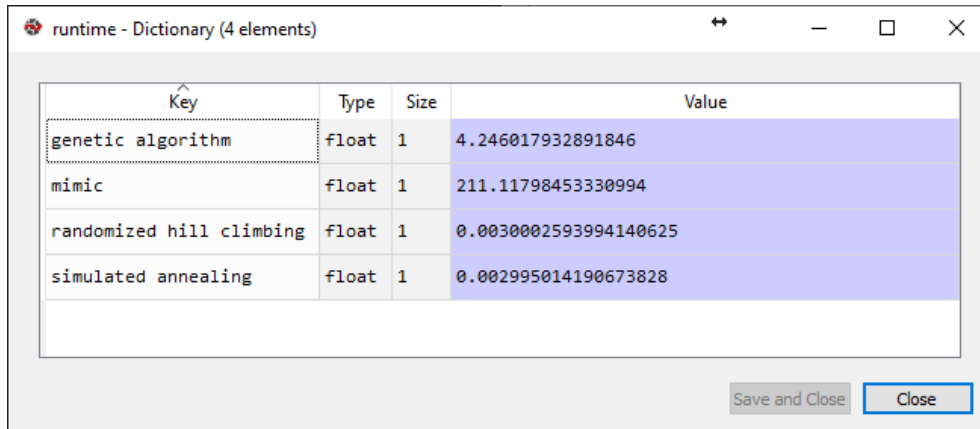
The Knapsack Problem is a classic NP-Hard optimization problem that every computer scientist has attempted to tackle during their algorithms course. There are many variants to the Knapsack problem, but for the purpose of simplicity we will be analyzing the discrete version of the knapsack problem, so we cannot take fractions of an item. For the knapsack problem, we used 81 different items and weights to simulate a complex problem to better differentiate which optimization algorithms will perform better for this problem. For the knapsack problem, the following parameters were used for each optimization algorithm:

Randomized Optimization Algorithm	Optimal Parameters
Genetic Algorithm	pop_size = 500, mutation_prob = 0.001, max_attempts = 500, max_iters = 100, N = 81

Simulated Annealing	<code>schedule = GeomDecay(), max_attempts = 500, max_iters = 100, N = 81</code>
MIMIC	<code>pop_size = 500, keep_pct = 0.8, max_attempts = 500, max_iters = 100, N = 81</code>
Randomized Hill Climbing	<code>max_attempts = 500, max_iters = 100, N = 81</code>

With the parameters specified above, my experiment yielded these results for the 81 item Knapsack Problem:





Key	Type	Size	Value
genetic algorithm	float	1	4.246017932891846
mimic	float	1	211.11798453330994
randomized hill climbing	float	1	0.0030002593994140625
simulated annealing	float	1	0.002995014190673828

The graph above for Fitness vs. Iterations clearly shows us that MIMIC performed the best, genetic algorithms came in second, randomized hill climbing came in third, and simulated annealing has the worst fitness score. For the knapsack problem, it's evident that the simulated annealing and randomized hill climbing algorithms are horrible choices for this problem because their fitness score does not show improvement as the number of iterations increases, however it is the complete opposite for the genetic algorithms and MIMIC as their fitness scores increase as the number of iterations increases. When analyzing the runtime for all four algorithms, it is obvious that randomized hill climbing and simulated annealing will finish a lot faster than the other two algorithms due to the fact that they are meant to finish faster per iteration. However, what is the point of finishing quickly when the results are not great? With genetic algorithms finishing in 4.25 seconds and MIMIC finishing in 211.12 seconds, we can easily move to genetic algorithms if timing is an issue because genetic algorithms provide decent results with a fast runtime. However, the best algorithm for this problem will be MIMIC because it shows the most improvement as the number of iterations increases. MIMIC takes longer to complete, but the time for completion is not absurdly high which makes it the best since it has a feasible running time with the best results. For such reasons, MIMIC is the preferred algorithm for the discrete knapsack because the problem is really complex and MIMIC works great for finding solutions for very complex problems but has the tradeoff of long runtimes.

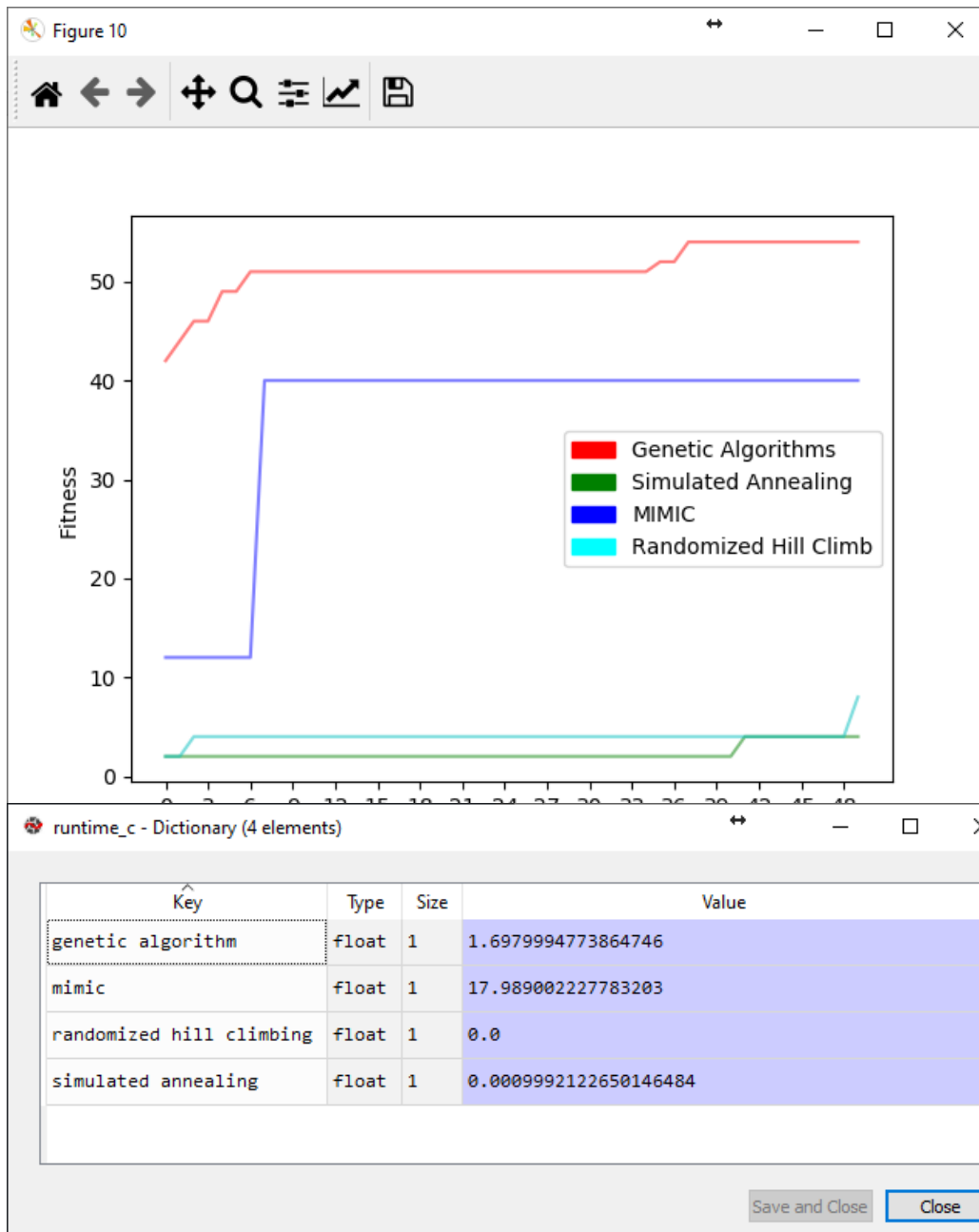
1.4 Four Peaks Problem (Genetic Algorithm)

The four peaks problem is a simple problem for humans where we are tasked to find the global maxima of 1's and 0's. However, the problem is difficult for optimization algorithms because the basin of attraction for the inferior local maxima gets larger and larger as we increase the size of the problem. We will be examining the four peaks problem using a $t_{pct} = 0.15$, where t_{pct} is a parameter define as the threshold parameter for the state space dimension. For the four peaks problem with an array of size 33, the following parameters were used for each optimization algorithm:

Randomized Optimization Algorithm	Optimal Parameters
Genetic Algorithm	pop_size = 500, mutation_prob = 0.001, max_attempts = 500, max_iters = 100, N = 33

Simulated Annealing	schedule = GeomDecay(), max_attempts = 500, max_iters = 100, N = 33
MIMIC	pop_size = 500, keep_pct = 0.8, max_attempts = 500, max_iters = 100, N = 33
Randomized Hill Climbing	max_attempts = 500, max_iters = 100, N = 33

With the parameters specified above, my experiment yielded these results for the Four Peaks Problem with an array of length 33:



have difficulty finding the global maxima once they find the local maxima. However, both genetic algorithms and mimic are great for finding the global maxima with genetic algorithm performing slightly better in this case. Examining the runtime for all four algorithms, we can conclude that the best algorithm for the four peaks problem will be the genetic algorithm because it has the best fitness score and the lowest runtime when compared to the mimic algorithm; randomized hill climbing and simulated annealing are not taken into consideration because they were stuck at the local maxima.

1.5 Conclusion

After finishing the analysis on the n-queens, knapsack, and four peaks problem, we can conclude that the randomized hill climbing and simulated annealing algorithms are much faster than mimic and genetic algorithms but have an issue of being stuck at the local maxima/minima. For simple problems and small basins of attractions, simulated annealing and randomized hill climbing perform well; in our case, for the n-queens problem. However, for more complex problems genetic algorithms and mimic will perform much better as seen in the knapsack problem but face an issue of long runtimes. To sum everything up, genetic algorithms and mimic are more likely to find the global maxima/minima but has a trade-off of having really long runtimes whereas simulated annealing and randomized hill climbing has short runtimes but has a trade-off of being stuck at the local maxima/optima.

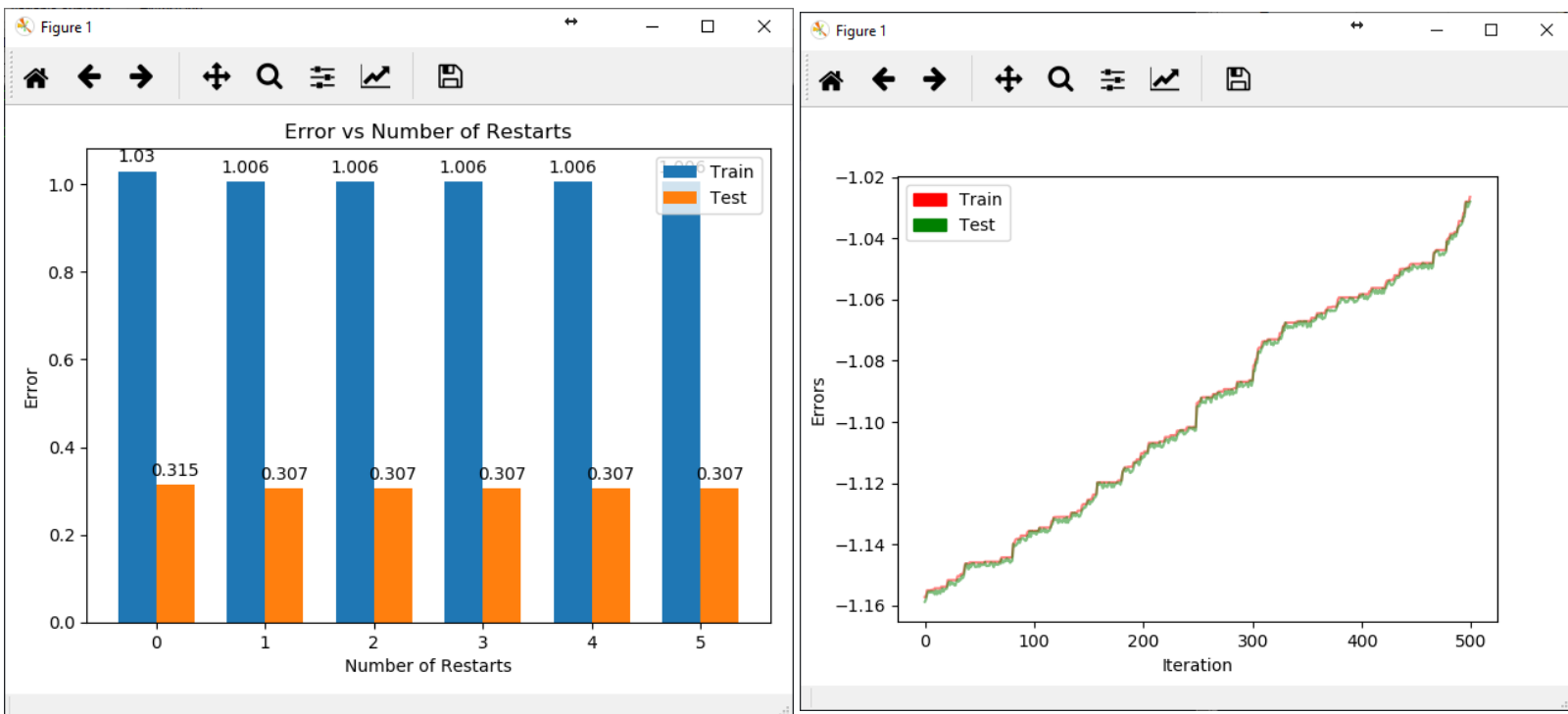
2 Randomized Optimization and Neural Network Weights

2.1 Introduction and Recap

In the Supervised Learning paper, we examined two datasets: the “Adult Data Set” from the UCI Machine Learning Repository and the “Graduate Admissions Data Set” from Kaggle”. We will be using the same neural network architecture from the previous assignment, but instead of back propagation with gradient descent, we will be using three randomized optimization algorithms for finding optimal weights: randomized hill climbing, simulated annealing, and genetic algorithms. The neural network used in the previous assignment consisted of two layers and eight neurons each layer. Therefore, for this problem we will be using a neural network with two layers and eight neurons to compare and contrast the three optimization algorithms with the back propagation gradient descent neural network from the previous assignment.

2.2 Randomized Hill Climbing (RHC)

Randomized hill climbing optimization is defined as the process of starting at a random point in space, examining the nearby points, move to the point with the best fitness, and then repeat itself. The algorithm proceeds until it cannot find a better fitness, and then it restarts itself. This process is repeated for as many restarts as the user specifies, which means that the only hyperparameter for the RHC algorithm is the number of restarts.

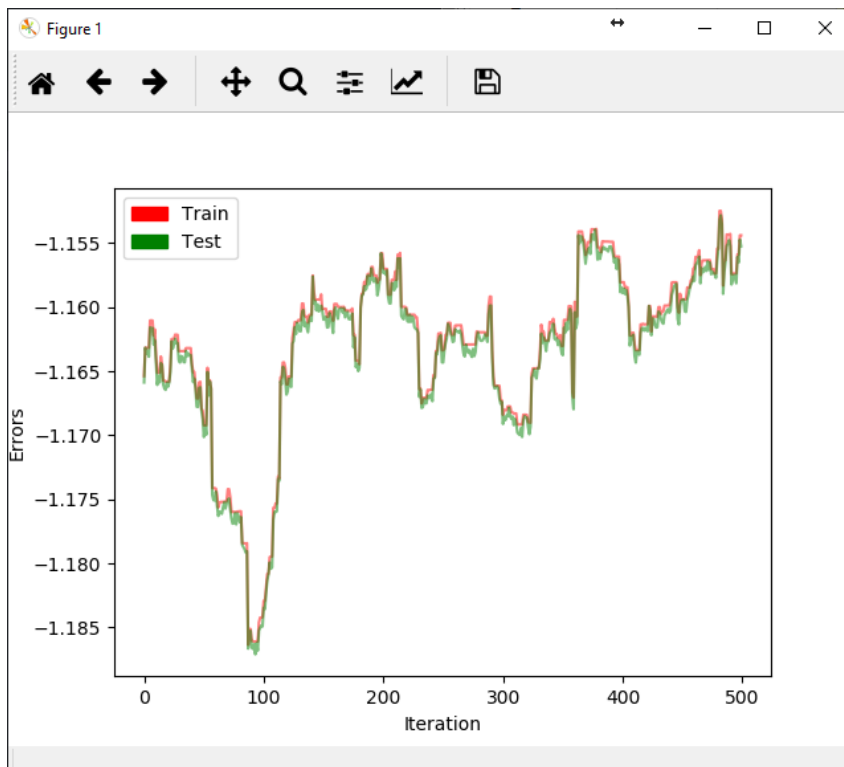


We can see that restarting the algorithm can provide an accuracy boost as shown above. However, for this particular problem the bar graph shows that the number of restarts doesn't matter after one, so that will be the optimal number of restarts for this problem. Running the neural net for 500 iterations on RHC shows that the error rate continues to decrease and still has potential for decreasing. However, my PC has issues and starts to freeze when the iteration count is too high which is holding me back from training for a longer period of time. Analysis of the Errors vs Iterations graph shows that the error rate for RHC on the neural network can be dropped even further. The optimal weights for this particular neural network may not be the best because RHC is known to get stuck at the local minimum. But, the

Error vs Number of Restarts bar graph shows that the number of restarts does not matter after one, so it is safe to assume that the result we obtain will be decent.

2.3 Simulated Annealing (SA)

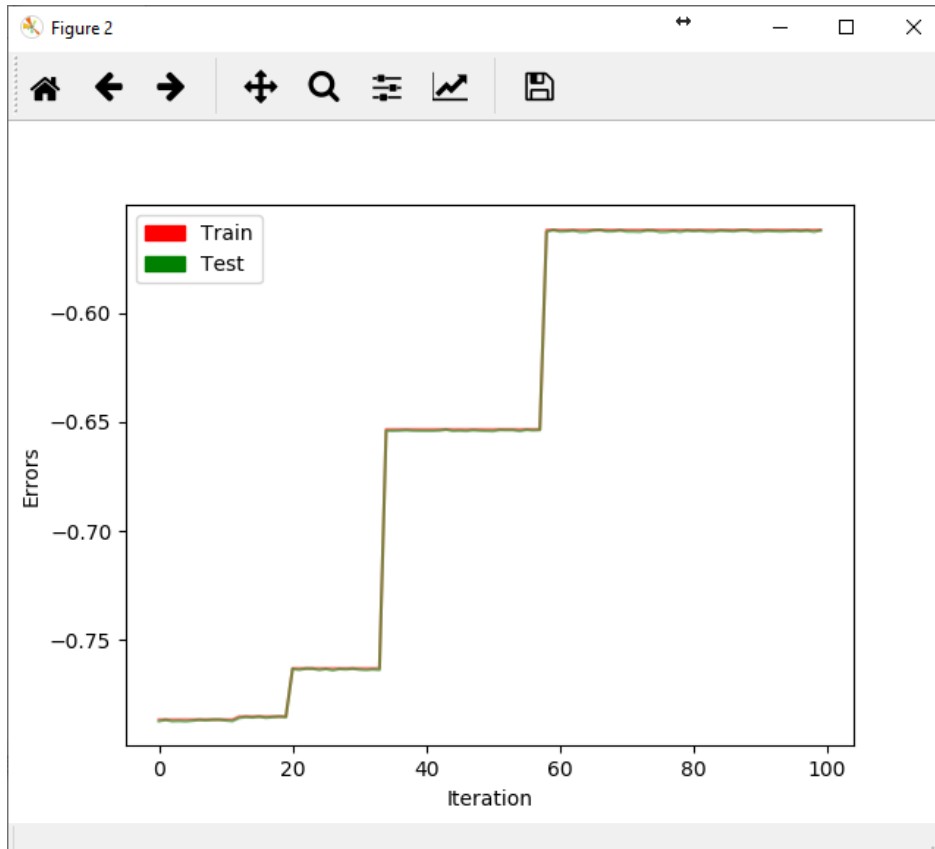
Simulated Annealing is a randomized optimization algorithm that also begins at a random point in space, then it begins by obtaining another random point in space. Then, the distance between the two points is calculated using the Boltzmann distribution. The algorithm then decides whether moving to this new point is better or not depending on the fitness function provided. This has the benefit of moving away from local minima/optima. However, this can backfire and we may end up moving to another local minima/optima.



We can see that using simulated annealing, the graph decreases its learning rate but not by a large amount. It oscillates back and forth between the error rate, but the average line is positive, towards a decreasing error rate.

2.4 Genetic Algorithms (GA)

Genetic Algorithms are an interesting family of algorithms and function as their name implies, genetic evolution. It is a process inspired by nature's evolution of species and natural selection. The algorithm works as follows: candidate locations that have low errors are "mated" together with a probability of mutation to hopefully obtain a better location with an even lower error rate. This works exceptionally well for finding the global minima; however, the process is very slow. The runtime for this algorithm is extremely slow per iteration, but converges in less iterations than simulated annealing and random hill climbing.



We can see that genetic algorithm improves over time and the error rate is extremely close to zero. Another observation is that the algorithm does not oscillate like simulated annealing; every iteration for genetic algorithm does better and better, and never does worse. Even though genetic algorithm looks great in the graph, the runtime for the algorithm is extremely long. Genetic algorithm took 50x longer than simulated annealing and random hill climbing on my PC.

2.5 Conclusion

In the end, it is evident that genetic algorithms perform better than simulated annealing and random hill climbing as shown by the error rate. Genetic algorithms achieved an error rate of 0.55 whereas simulated annealing achieved 1.155, and random hill climbing achieved 1.02. Even though genetic algorithm achieved the lowest error rate, there was still a drawback for this: the runtime. Genetic algorithms require more time to run per iteration, but has the benefit of requiring less iterations. As mentioned previously, my PC has an expected 50x longer runtime for genetic algorithms than simulated annealing and random hill climbing. To wrap everything up and conclude the analysis, genetic algorithms and mimic provides great accuracies that are close to the global optima but have a horrendously long runtime as a tradeoff; simulated annealing and random hill climbing provide decent accuracies but may end up being stuck in the local optima. In the end, the best optimization algorithm depends on how complex the problem is and whether the extra runtime is worth the accuracy boost.