Assignment 2 - Randomized Optimization

[*] This assignment is adapted from the report I submitted last semester.

Steven Dale
sdale30@gatech.edu

***Abstract*—In this paper, we explore the properties and behaviors of three randomized optimization algorithms: Random Hill Climbing (RHA), Simulated Annealing (SA), and Genetic Algorithms (GA). We structure our analysis into two main sections. The first section highlights the strengths of GA and SA on two distinct optimization problems. In the second section we use four methods (gradient descent, RHC, SA, and GA) to update the weights of a simple neural network for the MNIST classification task. In our analysis we gain insights into the strengths and weaknesses of each optimization method for a variety of problem types.**

****I apologize for parts of this assignment which are a bit messy or missing and make the report a bit hard to read. I spent far too long on my experiments and ran out of time to complete the report.**

I. Two Discrete Optimization Problems

A. Hypothesis

Hypothesis I - The maximum fitness of SA will be greater than or equal to the maximum fitness of RHC. In addition, SA will take less time to converge. This is because SA is equivalent to RHC for $Temperature = 0$. RHC does have an advantage with being able to randomly restart, but this will generally be computationally expensive and not as effective as the tempered exploration of SA.

Hypothesis II - The SA algorithm will perform best for algorithms that have a reasonably sized global optima. For these problems, the SA algorithm will stumble upon the global optima while the temperature is high. As the temperature decreases to zero, the SA algorithm will hill climb to the global optima. Conversely, the SA algorithm will perform poorly for problems with large basins of attraction and small global optima areas. The SA algorithm will struggle to find the small global optima area before the temperature cools. As the temperature approaches zero, it will hill climb to a local optima.

Hypothesis III - The GA algorithm will perform better than RHC and SA on problems with the following conditions:

1) the global optima is small (large basin of attraction)
2) the globally optimal state can be easily be discovered via the crossover of two locally optimal states

B. Methodology

Preliminary Investigation: For each problem, I performed a preliminary investigation wherein I ran each optimization method on a variety of parameter settings (problem size, max attempts, max iterations, etc.). By doing this, I was able to identify a configuration that would demonstrate interesting aspects of each optimization method, while still running in a reasonable amount of time.

Step 1 - Hyperparameter Tuning: In the first part of the investigation, I fixed the problem size (determined in step 0) and ran each method with a variety of hyperparameter settings. For RHC I varied the number of restarts, for SA I varied the initial temperature and the decay function, for GA I varied the population size and the mutation rate. For each method, I plotted the fitness curve (fitness vs. iterations) and tabulated the hyperparameter settings, maximum fitness, iterations, wall clock runtime, and function evaluation count.

Step 2 - Problem Size Analysis: Using the metrics from Step 1, I was able to identify hyperparameter values that produced the optimal fitness (using the minimum wall clock time to split ties). Using these parameter values, I ran each method again, this time using a variety of problem sizes while keeping the other hyperparameters fixed. I then plotted the fitness curve and runtime curve in the same figure. Overlaying both y-axes allowed me to see the impact of problem size on both fitness and runtime along side each other. This is important because there is a trade-off between fitness and runtime. In choosing the right algorithm for a problem, we must consider both metrics.

Handling Variability For each experiment, I ran the algorithm three times, each time with a different seed. This allows use to get an idea of the variability of each algorithm. Throughout this report we provide the mean metrics across all three runs.

C. Four Peaks Problem - (Adept for Genetic Algorithms)

Problem Description

Four Peaks is an optimization problem wherein a string of 0's and 1's in the input and we are trying to maximize the following reward function:

$$f(x) = max(head(1,x), tail(0)) + R(x) \tag{1}$$

Where:

- $head(1,x)$ is the number of consecutive 1s from the start of the string
- $tail(0,x)$ is the number of consecutive 0s from the end of the string
- $R(x)$ is a reward function that adds a bonus of $length(x)$ if both $head(1,x)$ and $tail(0,x)$ exceed the threshold, T
- $T_{\text{PCT}} = T \div Length(x)$

Note that T_{PCT} must be less than 0.5 to allow for the possibility of a bonus. This is because the string cannot contain more than half 0's and more than half 1's.

Four Peaks has a very interesting property that will dramatically affect the performance of our randomized optimization algorithms. The problem has a large basin of attraction and small global optima. The higher the value of T_{PCT}, the larger the basing of attraction is and the smaller the global optima area is. To demonstrate this property and its effects very clearly, I chose to use a very high value of $T_{\text{PCT}} = 0.4$ for all Four Peaks experiments.

Another interesting thing about the Four Peaks problem is that we know that all of the local optima are equal to the input size. That means that if our fitness score is less than the input size, then we are definitely not in a local optima. We will use this fact in our analysis.

For the initial hyperparameter tuning, I chose a problem size of 30. This means that the global optima will be a fitness of 47 and the local optima will be 30 (when the string is all 0's or 1's). Knowing the local optima will help us understand when our algorithms are likely falling victim to the local optima.

Note, we will use the normalized fitness scores. For reference the local optima score of 30 is 0.638 after normalization and the global optima score of 47 is 1.0 after normalization.

Four Peaks - Random Hill Climbing - Hyperparameter Tuning

Fixed Parameters:

- Problem Size = 30
- Max Attempts = 150
- Max Iterations = 10,000

Variable Parameters:

- Restarts = [1, 20, 50, 100, 200]

TABLE I
FOUR PEAKS - RANDOM HILL CLIMBING - RESTARTS

Restarts	Mean Fitness (norm)	Runtime (sec.)	FEvals	Iterations
1	0.545	0.14	567	383
20	0.624	11.54	6445	636
50	**0.638**	62.76	13749	777
100	0.638	308.95	29083	834
200	0.638	1345.20	44352	861

The RHC algorithm was never able to achieve the global optima due to it's inability to find very small global optima. The algorithm was able to consistently find the local optima at 0.638 with 50 or more restarts.

Interestingly, for two of the hyperparameter choices, Restarts = 1 and 20, the algorithm did not always converge on the local optima. We know this, because the score is less than 0.638 which is the only local optima score for this problem. The mean number of iterations is far less than the max of 10,000, so this only means that the algorithm stopped due to the max number of attempts.

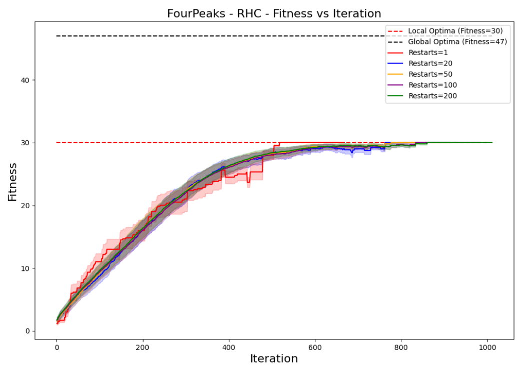

Fig. 1. Fitness curve of Random Hill Climbing on the Four Peaks problem using a range of restart values.

It turns out that for each sub-optimal state less than the local optima of 0.638 (when we haven't gotten the bonus), we can improve our score by changing the next leading 1 or trailing 0. No other change can increase our fitness. This means that only 1 neighbor will be able to improve our score. When I set the number of max attempts, I did not consider this. I figured that 150 attempts would be enough to cover a space of 30 neighbors. However, since the random selection of neighbors is a selection with replacement operation, it is still reasonably likely that the one correct neighbor will not be sampled and our algorithm will halt.

To fix this in future experiments I should either make the neighbor sampling "without replacement" or increase the max attempts to a much higher value.

Four Peaks - Simulated Annealing - Hyperparameter Tuning

Fixed Parameters:

- Problem Size = 30
- Max Attempts = 150
- Max Iterations = 10,000

Variable Parameters:

- Start Temperature = [1, 10, 100, 1000, 10000]
- Decay Function = [Arithmetic, Geometric, Exponential]

For each decay function, I chose to set a value of R that would ensure that the decay reaches it's minimum value when the algorithm reach 80% of the maximum iterations. This made it easy to compare the different decay functions as they would each reach the minimum before the last iteration. Why did I choose 80% instead of having it reach the minimum temperature at the last iteration? Well, I wanted the algorithm to get a chance to continue hill climbing after the temperature reached it's minimum. This turned out to be very important for the arithmetic decay function, which decreases slower than the geometric and exponential ones.

As a result of fixing the decay functions to reach their minimum at 80% iterations, I realized that the geometric and exponential functions are identical. Although they have different forms, it turns out that every geometric equation has an exactly equivalent exponential one. For this reason, all of my results for the geometric decay function are identical to the exponential one.

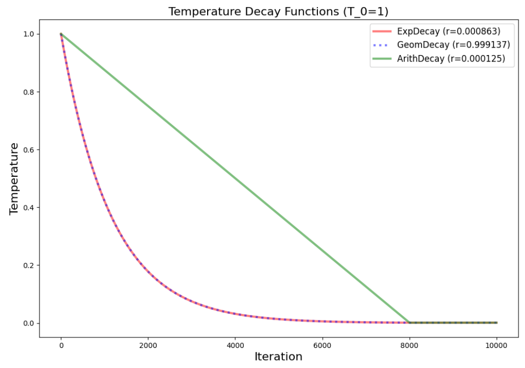

Fig. 2. Temperature Decay over 10,000 iterations with a start temperature of 1.

TABLE II
FOUR PEAKS - SIMULATED ANNEALING - TEMP. AND DECAY FN

Temp	Decay Fn	Mean Fitness (norm)	Runtime (sec.)	FEvals
1	Arith	0.638	8.32	1629
1	Exp	0.638	8.21	897
1	Geom	0.638	8.22	897
10	Arith	0.638	8.30	17933
10	Exp	0.638	8.16	5527
10	Geom	0.638	8.26	5527
100	Arith	0.255	8.37	19909
100	Exp	0.638	8.35	8513
100	Geom	0.638	8.24	8513
1000	Arith	0.149	8.35	13269
1000	Exp	0.638	8.36	10786
1000	Geom	0.638	8.38	10786
10000	Arith	0.128	8.43	15323
10000	Exp	0.638	8.31	11815
10000	Geom	0.638	8.19	11815

The mean fitness of almost every hyperparameter setting for SA converged on the local optima of 0.638. It appears that the SA algorithm also fell victim to the local optima problem with Four Peaks, just as RHC had. This is consistent with our understanding of SA, wherein, we can jump around when the temperature is high, allowing us to maybe find the global optima as we cool down. However, since the global optima area is so sparse for Four Peaks, even SA still has trouble finding it.

The experiments with Arithmetic decay functions and high starting temperatures did not converge on the local optima. This is likely due to them not having enough time with a low temperature when they could hill climb. Since they started with such a high temperature, and the cooling was linear (slow), it did not get a lot of iterations to do uninterupted hill climbing.

We see that the mean wall clock runtime for each SA hyperparameter setting was approximately 8 seconds. This is due to us setting the temperature decay function to reach it's minimum around 80% of the max iterations.

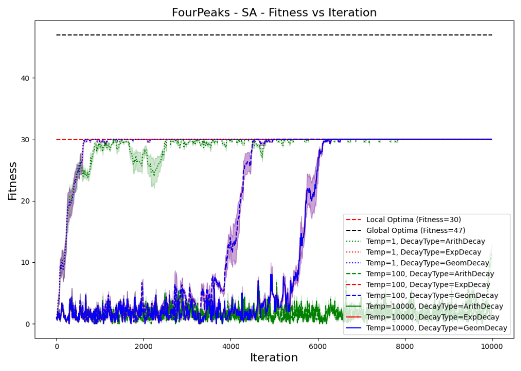

Fig. 3. Fitness curve of Simulated Annealing on the Four Peaks problem using a range of restart values.

The number of Function Evaluations for the Arithmetic decay function are greater than that of the other two functions because the Arithmentic decay takes much longer to reach small temperatures and therefore will take longer to converge on a local optima and terminate.

Four Peaks - Genetic Algorithm - Hyperparameter Tuning

Fixed Parameters:

- Problem Size = 30
- Max Attempts = 150
- Max Iterations = 10,000

Variable Parameters:

- Population Size = [10, 100, 500, 1000]
- Mutation Rate = [0.1, 0.3, 0.5, 0.7]

TABLE III
GENETIC ALGORITHMS - POPULATION SIZE AND MUTATION RATE

Pop. Size	Mut. Rate	Mean Fitness (norm)	Runtime (sec.)	FEvals
10	0.1	0.539	0.14	5157
10	0.3	0.638	0.09	3323
10	0.5	0.610	0.11	3347
10	0.7	0.638	0.07	1927
100	0.1	0.638	0.38	7787
100	0.3	0.638	0.34	3544
100	0.5	0.638	0.36	4757
100	0.7	0.759	0.35	3645
500	0.1	1.000	1.83	20049
500	0.3	0.879	1.78	14536
500	0.5	0.986	1.78	11529
500	0.7	0.759	1.92	15541
1000	0.1	1.000	3.83	17023
1000	0.3	1.000	4.89	126132
1000	0.5	0.879	4.11	23030
1000	0.7	1.000	4.12	19025

The genetic algorithm was the only algorithm to reach the optimal fitness value of 1.0. It was able to do this for

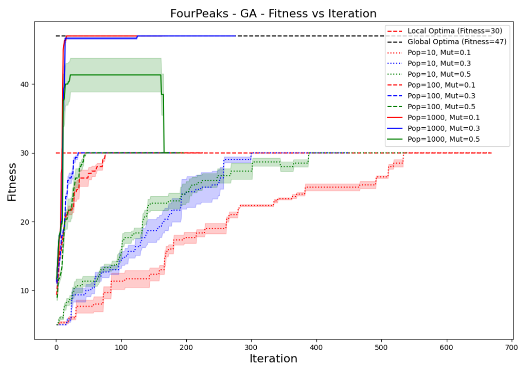

Fig. 4. Fitness curve of Genetic Algorithms on the Four Peaks problem using a range of restart values.

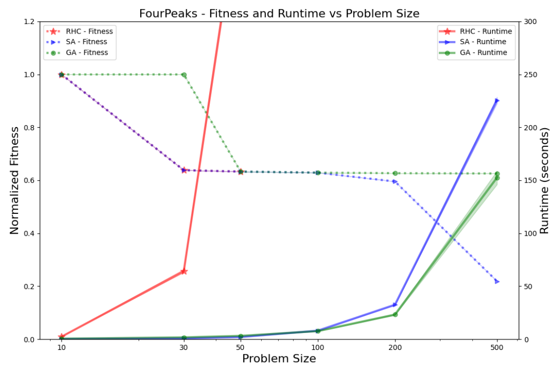

Fig. 5. Fitness and Runtime vs Problem Size for Four Peaks problem.

population sizes of 1,000 or greater and a mutation rate of 0.1 or 0.3. This is consistent with Hypothesis III. Genetic algorithms performed so well on this problem due to the nature of the problem. The advantage of this method is it's ability to meld together two good states and make an even better state with them. The structure of this problem, having ones in the front and zeros in the back makes the bit-wise mating of two states very effective. Consider Four Peaks problem of size 5, we have two sub-optimal states which are stuck in local optima: "11111" and "00000". When we combine these two states it's possible that we get something like "10100", which would not be the global optima, but would shake us out of the local optima we were stuck in before. After that, we might mutate the second bit and suddenly we're in the global optima "11100".

It appears that a population size of only 10 or 100 was not enough samples to successfully find the global optima. However, many of them did converge on the local optima of 0.638. This indicates that our population diversity might have collapsed. For example, if all 10 of our samples are "11111" or close to that, then we are very unlikely to get anymore zeros from breeding states and we will be stuck.

Problem Size Analysis

Figure 5 shows an analysis of fitness and runtime for each algorithm's best configuration of hyperparameters.

The curve for RHC stops at the problem size of 50, this is because RHC increased dramatically in runtime as the problem size grew and running it was no longer tenable. This is partially due to the use of Restarts=200 for RHC which performs best, but takes the longest. For the smaller problem sizes of 10, 30, and 50, RHC was able to match the performance of SA which had optimal fitness for problem size = 10 and locally optimal performance for problem sizes 30 and 50.

The curve for SA shows that the algorithm converges on the local optima when the state space becomes enormous. SA is able to converge on the global optima only for problem size = 10.

The genetic algorithm was able to reach the global optima for problem sizes of 10 and 30, but fell to the local optima for larger sizes. This indicates that we would need more resources to run GA on large problem sizes of Four Peaks. A larger population would be needed to capture the increased diversity of the larger state space.

D. *N-Queens Problem - (Adept for Simulated Annealing)*

Problem Description

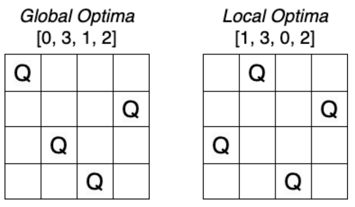

Fig. 6. Two examples of N-Queens states. The state on the left shows a global optima, with no queens in conflict. The state on the right shows a local optima, with one pair of queens in conflict and no neighboring state with a higher fitness.

N-Queens is an optimization problem wherein we are trying to position N queens on a $N \times N$ chess board so that there are the fewest pairs of queens attacking each other.

The input of the evaluation function is a list of column positions for each queen. It is assumed that the first queen is always on the first row, the second queen is always on the second row, and so on. The column position of the n$^{\text{th}}$ queen is designated by the n$^{\text{th}}$ integer in the input.

This problem is very interesting to study for randomized optimization algorithms because the problem becomes very complex as the problem size grows, having a large number of local optima and plateaus, and a sparse amount of global optima spread throughout the landscape.

N-Queens - Preliminary Investigation

The preliminary investigation for the N-Queens indicated that a problem size of 100 would be complex enough to demonstrate the effectiveness of SA over RHC and GA.

I used different max iteration values for each method. For the RHC and GA I used a max iterations of 1,000. For SA, which runs less FEvals per Iteration, I used 10,000 max iterations.

N-Queens - Random Hill Climbing - Hyperparameter Tuning

Fixed Parameters:

- Problem Size = 100
- Max Attempts = 500
- Max Iterations = 1,000

Variable Parameters:

- Restarts = [1, 5, 10, 20, 50]

TABLE IV
RANDOM HILL CLIMBING - RESTARTS

Restarts	Fitness (norm)	Runtime (sec.)	FEvals	Iterations
1	0.899	**1.40**	**966**	**632**
5	0.919	5.34	3187	849
10	0.918	12.07	4234	849
20	0.919	33.23	8441	898
50	**0.921**	152.52	17395	984

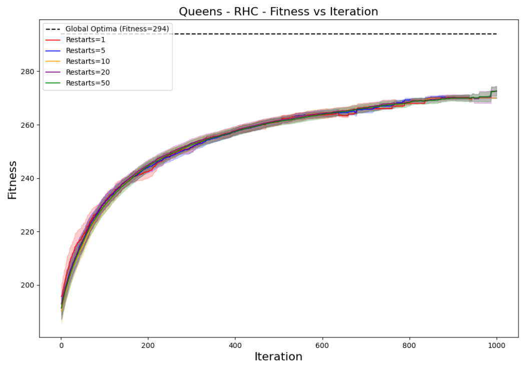

Fig. 7. Fitness curve of Random Hill Climbing on the N-Queens problem using a range of restart values.

Table I and Figure 2 show the performance of each RHC model with various restart values. It appears that the number of restarts does not impact the maximum fitness. This suggests that the RHC algorithm is falling into the same local optima each time its run. We know this because even when there is only 1 restart, we still reach the same maximum fitness score.

N-Queens - Simulated Annealing - Hyperparameter Tuning

Fixed Parameters:

- Problem Size = 100
- Max Attempts = 500
- Max Iterations = 10,000

Variable Parameters:

- Start Temperature = [0.01, 0.1, 1, 10, 100]
- Decay Function = [Arithmetic, Geometric, Exponential]

TABLE V
SIMULATED ANNEALING - TEMPERATURE AND DECAY FN

Temp	Decay Fn	Fitness (norm)	Runtime (sec.)	FEvals
0.01	Arith	**0.985**	15.89	9431
0.01	Exp	**0.985**	**15.76**	9431
0.01	Geom	**0.985**	15.77	9431
0.1	Arith	**0.985**	16.30	9431
0.1	Exp	**0.985**	15.95	9431
0.1	Geom	**0.985**	16.05	9431
1	Arith	0.971	17.85	11574
1	Exp	0.984	16.62	9482
1	Geom	0.984	16.93	9482
10	Arith	0.968	22.17	16478
10	Exp	**0.985**	17.60	12039
10	Geom	**0.985**	17.39	12039
100	Arith	0.951	22.07	17828
100	Exp	0.972	18.71	12409
100	Geom	0.972	18.64	12409

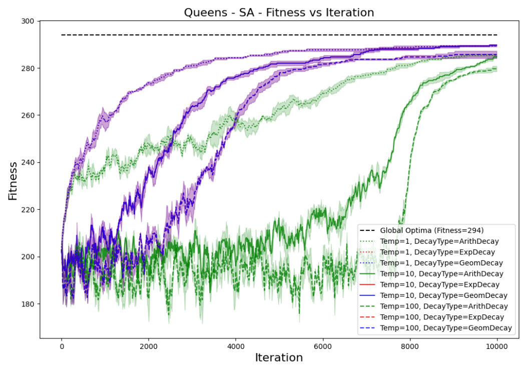

Fig. 8. Fitness curve of Simulated Annealing on the N-Queens problem using a range of temperature and decay values.

As with the Four Peaks problem, I used R values for each decay function that would reduce the temperature to its minimum at 80% of max iterations.

It appears that all temperatures and decay functions performed well, with each reaching greater than 0.95 norm fitness. However, the geometric and exponential decay functions worked best indicating that moving to lower temperatures faster is advantageous (i.e. exploring for less time and exploiting for more time is better). It also appears that lower temperature lead to faster convergence. This also supports the claim that only a small amount of exploration is necessary.

N-Queens - Genetic Algorithms - Hyperparameter Tuning

Fixed Parameters:

- Problem Size = 100
- Max Attempts = 500
- Max Iterations = 1,000

Variable Parameters:

- Population Size = [10, 20, 50, 100]
- Mutation Rate = [0.1, 0.3, 0.5, 0.7]

TABLE VI
GENETIC ALGORITHMS - POPULATION SIZE AND MUTATION RATE

Pop. Size	Mut. Rate	Fitness (norm)	Runtime (sec.)	FEvals
10	0.10	0.867	5.01	5556
10	0.30	0.905	4.82	4308
10	0.50	0.917	**4.66**	5093
10	0.70	0.909	5.41	**4155**
20	0.10	0.901	12.16	10918
20	0.30	0.938	11.01	11237
20	0.50	0.935	11.09	9598
20	0.70	0.956	13.22	12753
50	0.10	0.922	23.94	15135
50	0.30	0.964	23.75	27128
50	0.50	0.964	23.54	26461
50	0.70	0.957	23.97	21361
100	0.10	0.949	32.67	32561
100	0.30	0.968	39.20	38927
100	0.50	0.969	36.10	33880
100	0.70	**0.977**	45.53	44685

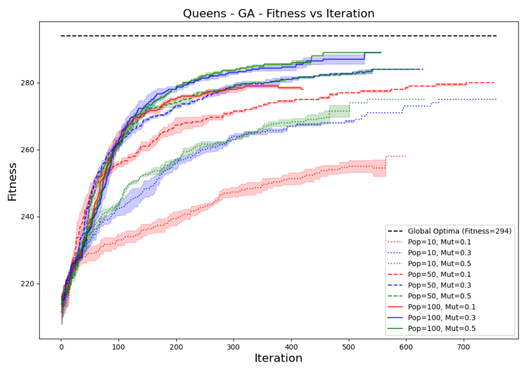

Fig. 9. Fitness curve of Genetic Algorithms on the N-Queens problem using a range of population size and mutation rate values.

The max fitness for GA appears to increase with the size of the population. For GA, the initial randomly selected population is the main form of exploration (breeding and mutating states are weak exploration because they are small/biased changes). The N-Queens problem has many local optima and sparse global optima, for this reason, the exploration by having a large population size is critically important.

However, GA is an expensive algorithm that gets more expensive with increased population size. I had to cut off the max population to 100 so that tests could be run in a feasible amount of time. We see that a population size of 100 requires 30-40 seconds of runtime. With simulated annealing reaching higher fitness in only 15-20 seconds, I did not feel that testing larger population sizes that take even longer would be a fair comparison to the fast running SA algorithm.

N-Queens - Problem Size Analysis

Simulated Annealing attains higher fitness scores across all problem sizes except for the largest size, 200. GA attains a slightly higher fitness than SA at problem size 200, however, GA uses twice as much runtime for this. If we allowed SA a larger number of iterations, we would likely get a better fitness score than GA.

In addition to having a higher fitness score, the runtime of SA appears to grow much slower than both RHC and GA.

Simulated Annealing performs well on problems like N-Queens because it can easily escape local minima. In N-Queens there are many local minima and small random changes to the state can get us out of these local minima quickly. For this reason, SA is able to explore many different local optima and landing often on some of the higher local optima.

Notice, however, that none of the algorithms reach the optimal fitness score for N-Queens. This demonstrates the difficulty of this problem, wherein the optimal solutions are few and sparse.

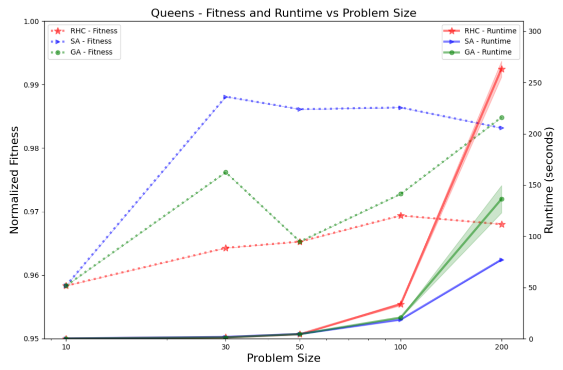

Fig. 10. TODO: caption

II. TRAINING NEURAL NETWORKS WITH RANDOMIZED OPTIMIZATION

In this section we look at the effect of training a neural network on the MNIST dataset using gradient descent, RHC, SA, and GA.

A. Dataset

The MNIST dataset is a collection of handwritten digits capture in 28x28 greyscale images. The goal of this problem is to classify which of the digits is drawn.

To reduce training/evaluation time, I resized each image to 14x14 and reduced the size of the dataset to 10,000 images.

B. Methodology

For my network I used a simple feed forward network with one hidden layer of size 10. This network is very small which makes it relatively efficient to run experiments on, but susceptible to bias (variance/bias trade-off).

For all results I used 5-fold cross validation.

For each experiment, I wanted to see the test accuracy in addition to the fitness as the model trained. This would help me determine when the model has over-fit or not. However, there was no way to do this with the mlrose library. So I decided to run each experiment over and over, each time using a slightly larger max iteration value. After each training, I would record the test accuracy of the model and then run the next increment. For example, if I wanted to train a NN with SA for 1,000 iterations, then I would train the model for 100 iterations, record the test accuracy, then train it for 200 iterations, record the test accuracy, then for 300, and so on until 1,000 iterations. This is a very inefficient way to implement validation curves because we are redoing so much computation each time (on the order of O(n^2). However, it got me my validation curves in the end.

For each randomized optimization algorithm, I first tested several values of learning rates while holding the other hyperparameters constant (I used my best naive guesses for the hyperparameter values). After doing this first step, I was able to identify the best learning rate to use. With this, I test out a variety of hyperparameter ranges using the optimal learning rate I had discovered.

There are two plot for each method, RHC, SA, and RHC. There is one plot for gradient descent which did not have any hyperparameters to tune other than learning rate.

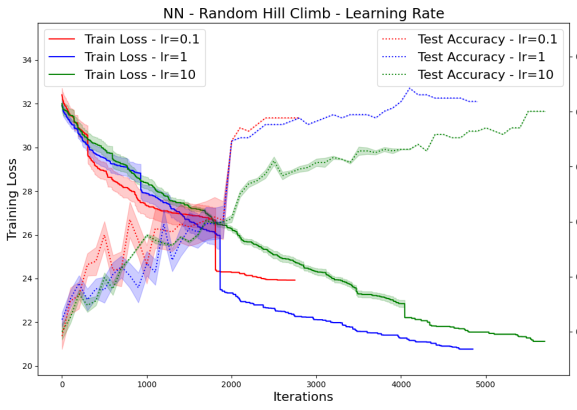

Fig. 11. Learning curve of RHC with various learning rates.

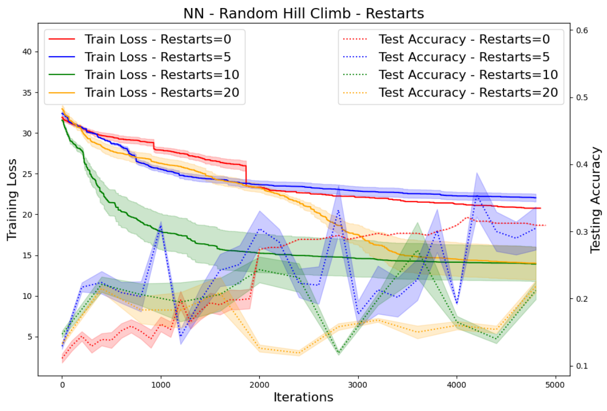

Fig. 12. Learning curve of RHC with various restart values.

RHC achieved its best performance of 35.5% test accuracy with the following parameters:

- Learning Rate: 1
- Restarts: 5

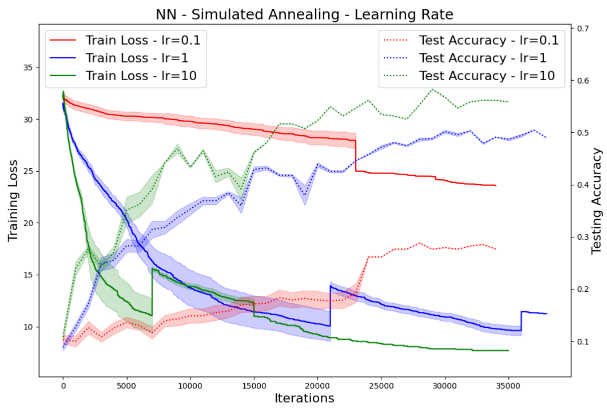

Fig. 13. Learning curve of SA with various learning rates.

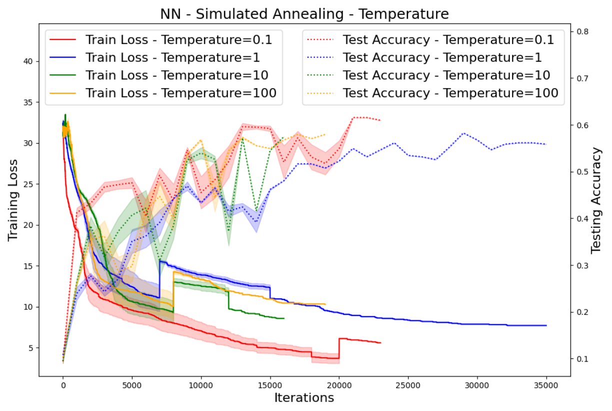

Fig. 14. Learning curve of SA with various start temperatures.

TABLE VII
NN - Simulated Annealing Performance

Learning Rate	Temp.	Test Accuracy	Runtime (sec.)	Iterations
0.1	10.0	61.6%	1961.11	23001
1	10.0	58.3%	8971.76	35001
10	10.0	57.5%	1289.76	16001
100	10.0	58.0%	1480.91	19001

SA achieved it's best performance of 61.6% test accuracy with the following parameters:

- Learning Rate: 10
- Temperature: 0.1

GA achieved it's best performance of 37.4% test accuracy with the following parameters:

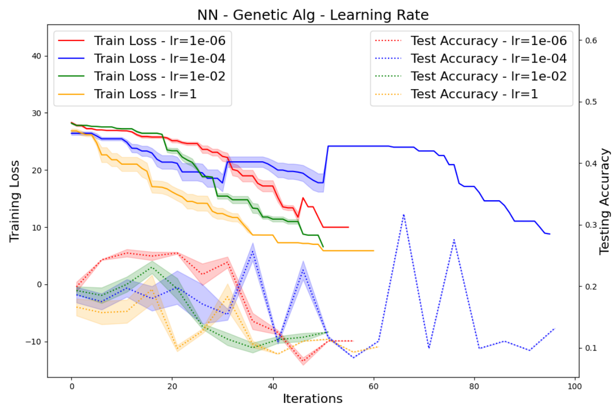

Fig. 15. Learning curve of GA with various learning rates.

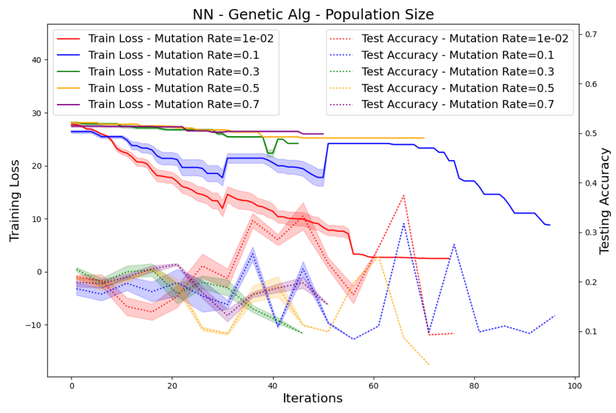

Fig. 16. Learning curve of GA with various mutation rates.

- Learning Rate: 1e-4
- Mutation Rate: 1e-2

TABLE VIII
NN - GRADIENT DESCENT PERFORMANCE

Learning Rate	Test Accuracy	Runtime (sec.)	Iterations
1e-7	30.8%	1532	3401
1e-8	27.0%	2506	4101
1e-9	15.6%	3296	6301

Gradient descent achieved it's best performance of 30.8% test accuracy with a learning rate of 1e-7.

C. Conclusion

Simulated Annealing achieves the highest test accuracy of all the methods, even then gradient descent.

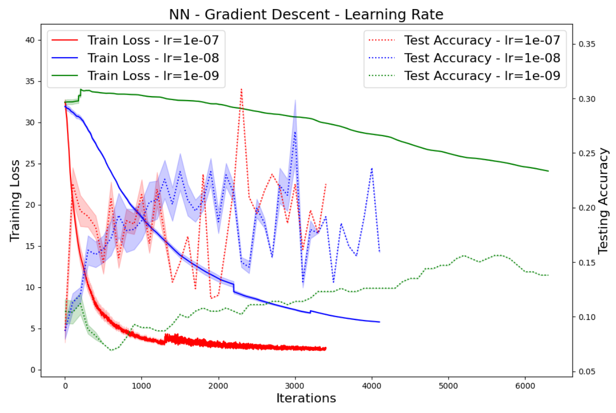

Fig. 17. Learning curve of gradient descent with various learning rates.