CS 7641 Assignment #2: Randomized Optimization

William Koppelman
wkoppelman3@gatech.edu

I. Introduction

In this paper I will explore four randomized optimization methods: randomized hill climbing, simulated annealing, a genetic algorithm, and MIMIC. I will use the first three of these methods to complete a local random search for the neural network weights using my model completed in assignment #1 for a red wine dataset. For the second part, I will optimize three fitness functions using all four methods to ascertain the strengths and weaknesses of each method.

II. Local Random Search for Neural Network Weights

A. *Backpropogation Benchmark from Assignment 1*

As a reminder, the red wine dataset is a collection of physicochemical attributes on the ordinal sensory output of quality. It only looked at Portuguese "Vinho Verde" red wine. There are 11 attributes and 1,599 instances in this dataset. The labels are a quality score that ranges from 0 to 10 but only scores between 3 and 8 are present in the data. More information can be found about the dataset in the README.

The neural network model created in assignment #1 used a backpropogation of a gradient descent optimizer to learn the weights. The model had three hidden layers with eight neurons each using a softplus activation function. I will use this same setup to compare the random optimization methods.

This network resulted in a training accuracy of 0.613, a test accuracy of 0.553. It had a mean fit time of 26.2831 seconds and a mean score time of 0.7943 seconds. These will be the benchmarks when evaluating the random optimization methods.

All programming was done is Python 3.x using the mlrose package (Mlrose.readthedocs.io, 2019). Using some modifications to this package allowed me to implement the same model described above, with the exception of the genetic algorithm for neural network weights.

B. *Randomized Hill Climbing*

The neural network weight optimization was used from the mlrose package to determine the appropriate weights for our pre-determined neural network. I used randomized hill climbing and varying the learning rates on a log-linear scale [0.001, 1] and using 5,000 iterations for my search space with an early stopping after 100 iterations without improvement. I was able to find that the learning rate of 0.8 provided the highest accuracy for the red wine dataset.

As seen if Fig. 1 the local search converged to its highest accuracy around 2,500 iterations.

Table I shows the optimized model provides a training accuracy of 0.605 and test accuracy of 0.556. It had a mean fit time of 10.7771 seconds and a mean score time of 0.003 seconds. This is comparable to our benchmark and faster.

Fig. 2 shows the learning curve for this model, it shows that the data has a strong convergence and the bias/variance tradeoff is close to optimal.

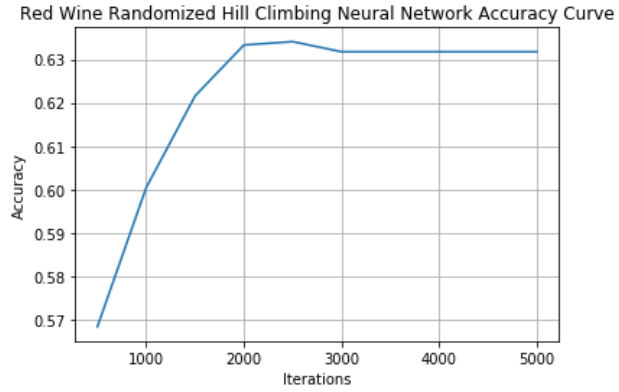

Fig. 1. Red wine neural network random hill climbing training accuracy vs. iterations

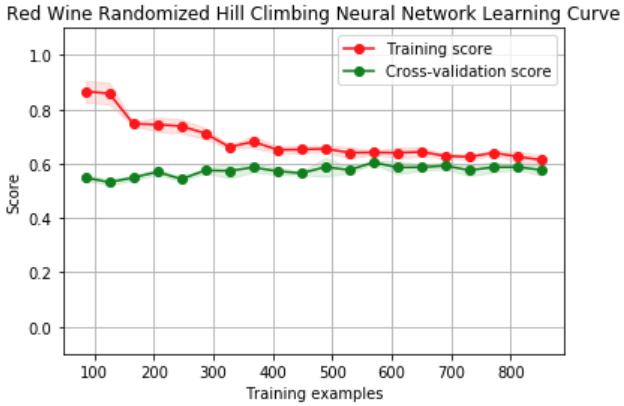

Fig. 2. Red wine neural network random hill climbing learning curve

TABLE I. STATISTICS OF LOCAL RANDOM SEARCH FOR RED WINE NEURAL NETWORK WEIGHTS

Model	Training Accuracy	Testing Accuracy	Training Time	Testing Time
Random Hill Climbing	0.605	0.556	10.7771	0.003
Simulated Annealing	0.599	0.541	33.8953	0.003
Genetic Algorithm	0.480	0.384	120.9398	0.0013

C. *Simulated Annealing*

When using simulated annealing to estimate the weights of the neural network I varied the temperature on a log-linear scale [0.001, 1] as well as its decay. I again used 5,000 iterations for my search space with an early stopping after 100 iterations without improvement. I was able to find that the temperature of 0.12 combined with a geometric decay provided the highest accuracy for the red wine dataset.

As seen if Fig. 3 the local search converged to its highest accuracy around 3,100 iterations.

Table I shows the optimized model provides a training accuracy of 0.599 and test accuracy of 0.541. It had a mean fit time of 33.8953 seconds and a mean score time of 0.003 seconds. This is slightly worse than our benchmark and about the same speed.

Fig. 4 shows the learning curve for this model, it again shows that the data has a strong convergence and the bias/variance tradeoff is close to optimal.

D. *Genetic Algorithm*

For the genetic algorithm I used populations varying from [100, 400] and mutation probabilities varying from [0.05, 0.30]. I again used 5,000 iterations for my search space with an early stopping after 100 iterations without improvement. I was able to find that the population of 400 and mutation probability of 0.1 provided the highest accuracy for the red wine dataset.

As seen if Fig. 5 the local search converged to its highest accuracy at the start and does not perform better the more we iterate.

Table I shows the optimized model provides a training accuracy of 0.480 and test accuracy of 0.384. It had a mean fit time of 120.9398 seconds and a mean score time of 0.0013

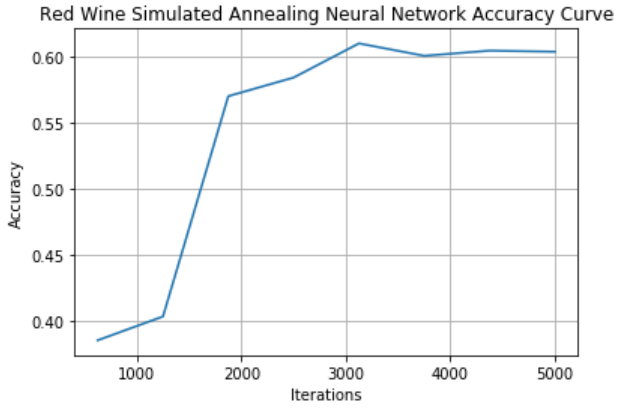

Fig. 5. Red wine neural network simulated annealing training accuracy vs. iterations

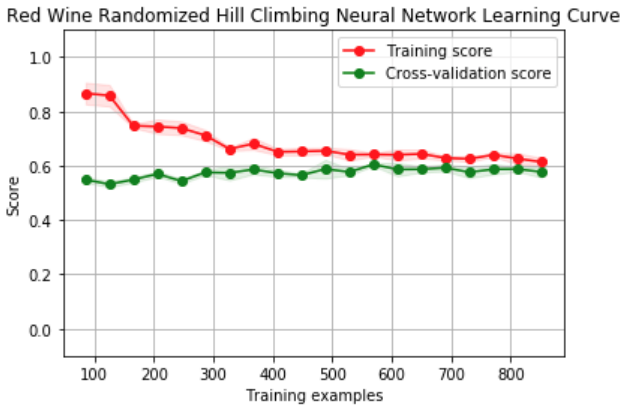

Fig. 6. Red wine neural network simulated annealing learning curve

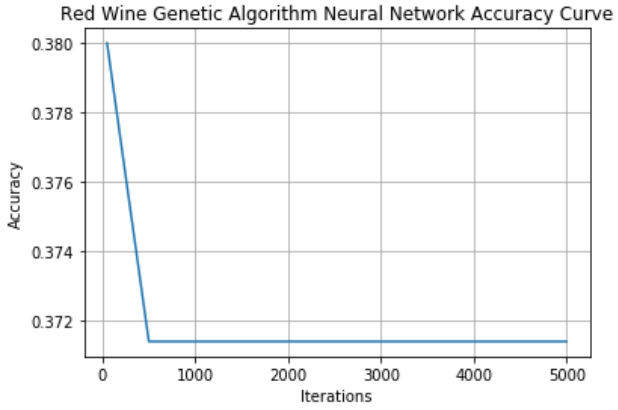

Fig. 3. Red wine neural network genetic algorithm training accuracy vs. iterations

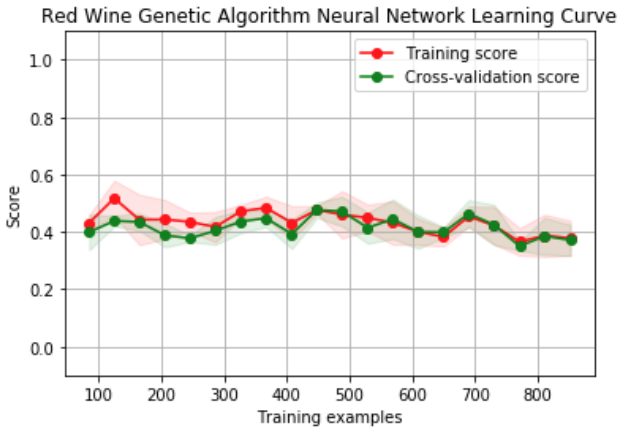

Fig. 4. Red wine neural network genetic algorithm learning curve

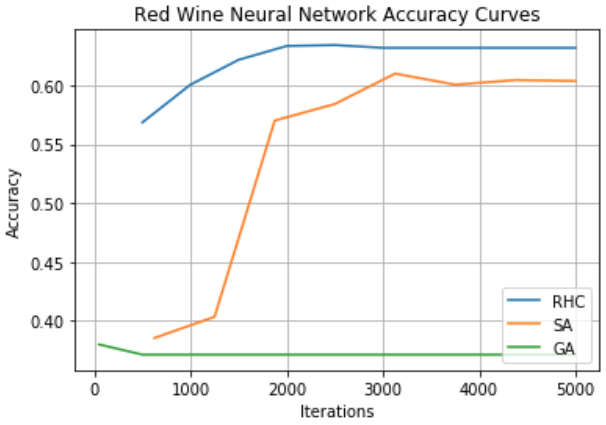

Fig. 8. Red wine neural network all algorithm training accuracy vs. iterations

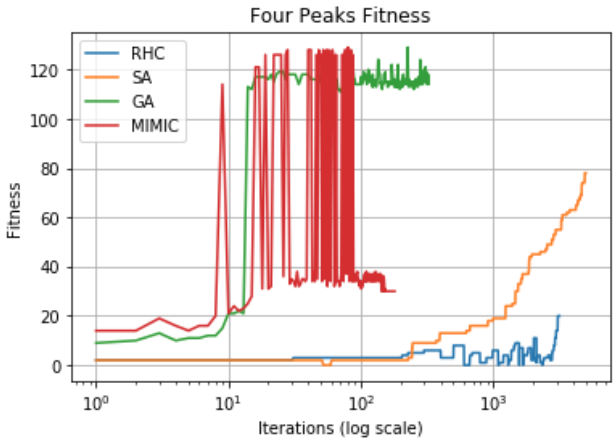

Fig. 7. Four Peaks comparison of all four algorithms fitness vs. iterations (log scale)

seconds. This is much worse than our benchmark and much slower as well.

Fig. 6 shows the learning curve for this model, it shows that the data contains a lot of random effects from a genetic algorithm that does not fit it very well. It does not overfit the training data, in fact it fits the training and validation equally poorly.

E. Comparison

Using these local search methods to find weights for our neural networks can be greatly affected by the complexity of our model. Since our weights are in a continuous domain, there are infinitely possible weights. Where in some domains, a sufficiently large amount of iterations will converge to a local optima, here we cannot guarantee that.

In our case, both simulated annealing and random hill climbing performed significantly better than the genetic algorithm (Fig 7). In fact, they performed similar to our gradient decent, backpropogation method.

The random hill climbing may have performed best because it is traveling along a continuous domain of our differentiable activation function and easily reaches a local optima. Simulated annealing and the genetic algorithm use randomness to jump around hoping to land in a better space. However, if there are infinitely many points to jump to, the probability of them finding a good one is essentially zero.

III. Local Random Search over Optimization Domains

A. Four Peaks

The four peaks problem is defined by (Mlrose.readthedocs.io, 2019):

$$Fitness(x,T) = \max\big(tail(0,x), head(1,x)\big) + R(x,T)$$

Where:

$tail(0,x)$ is the number of trailing 0's in x,

$head(1,x)$ is the number of leading 0's in x,

$R(x,T) = n$ if $tail(0,x) > T$ and $head(1,x) > T$ and

$R(x,T) = 0$ otherwise

This creates a reward based on a threshold T that are two optimal peaks in the fitness space, one each for $tail(0,x) > T$ and $head(1,x) > T$. There are also two sub-optimal peaks for each of $tail(0,x)$ and $head(1,x)$ when they are less than the threshold T. These two sub-optimal peaks become exponentially hard for hill-climbing to find as T increases (Baluja and Davies, 2019).

In this problem we expect the genetic algorithm and MIMIC to perform better than both random hill climbing and simulated annealing. I optimized this problem in mlrose with $n = 100$ and $T = 0.1$. Having a lower T will allow these two search methods to occasionally be successful. I ran all 4 algorithms over 5,000 iterations. I implemented early stopping if fitness was not improved over 100 iterations.

Table II shows a summary of the statistics for each search method. For random hill climbing I optimized the number of restarts. The algorithm converged to a fitness of 20 with 16 restarts after approximately 3,100 iterations.

For simulated annealing I optimized the decay schedule of the temperature parameter. The algorithm converged to a fitness of 78 with an exponential decay of the temperature after approximately 4,900 iterations.

TABLE II. Statistics of Local Random Search of Four Peaks Fitness Function

Model	Best Fitness	Iterations	Search Time (sec.)
Random Hill Climbing	20	3100	0.2081
Simulated Annealing	78	4900	0.6982
Genetic Algorithm	129	225	12.6575
MIMIC	129	80	1133.862

For the genetic algorithm I optimized the population size and mutation rate. With a population size of 150 and a mutation rate of 0.08 the algorithm's best fitness was 129 after approximately 225 iterations.

On the MIMIC algorithm I optimized the population size and the percentage to keep. The algorithm converged to a fitness of 129 with a population size of 400 and a 35% keep percentage after 80 iterations.

Fig 8 shows the comparison of each algorithm's fitness over the number of iterations. MIMIC and the genetic algorithm both found the largest fitness of 129 however, the genetic algorithm was two orders of magnitude faster at finding it.

You can see the large jump in the genetic algorithm's fitness that allows it to find the optimal peak. This is in large part to its crossover mutation that allows it to jump around.

MIMIC will clearly do well since it takes into account the structure of the domain space. However, the time it takes to find the optimal fitness is not ideal in this situation.

The random hill climbing does poorly, as expected. However the simulated annealing is able to find a much larger fitness state than the random hill climbing. Its temperature function no doubt helps to keep its search random enough to not get stuck in a suboptimal basin.

B. *One Max*

The one max problem, also known as count ones, maximizes the fitness of an *n*-dimensional vector as (Mlrose.readthedocs.io, 2019):

$$Fitness(x) = \sum_i x_i \quad \forall\, i \in x$$

This problem was evaluated as a discrete optimization, bit string problem. It has one global maximum of all 1's. Ideally randomized hill climbing and simulated annealing should perform well on this, since there is only one global maximum.

I optimized the one max problem in mlrose with $n = 100$. I ran all 4 algorithms over 5,000 iterations. I implemented early stopping if fitness was not improved over 100 iterations.

Table III shows a summary of the statistics for each search method. For random hill climbing I optimized the number of restarts. The algorithm converged to the optimal fitness with one restart after approximately 790 iterations.

TABLE III. STATISTICS OF LOCAL RANDOM SEARCH OF ONE MAX FITNESS FUNCTION

Model	Best Fitness	Iterations	Search Time (sec.)
Random Hill Climbing	100	790	0.048
Simulated Annealing	100	460	0.043
Genetic Algorithm	80	45	3.9902
MIMIC	100	15	801.1187

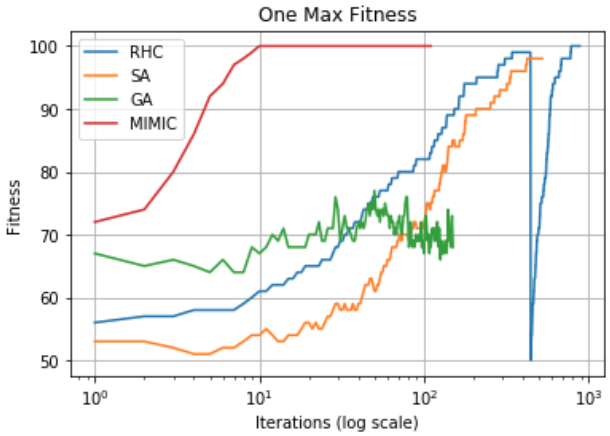

Fig. 9. One max comparison of all four algorithms fitness vs. iterations (log scale)

For simulated annealing I optimized the decay schedule of the temperature parameter. The algorithm converged to the optimal fitness with a geometric decay of the temperature after approximately 460 iterations.

For the genetic algorithm I optimized the population size and mutation rate. The algorithm did not find the optimal state. With a population size of 150 and a mutation rate of 0.02 the algorithm's best fitness was 80 after approximately 45 iterations.

On the MIMIC algorithm I optimized the population size and the percentage to keep. The algorithm converged to the optimal fitness with a population size of 200 and a 15% keep percentage after 15 iterations.

Fig 9 shows the comparison of each algorithm's fitness over the number of iterations. Clearly MIMIC was able to find the optimal state in the fewest iterations but the speed of randomized hill climbing and simulated annealing more than make up for the iterations. The additional computation time that MIMIC needs to shrink the hypothesis space is unnecessary for this problem.

As said before, this is a simple one optimum problem that is ideal for random hill climbing and simulated annealing. Simulated annealing was able to find the optimal solution in fewer iterations than random hill climbing thanks to its temperature parameter.

I believe the reason that the genetic algorithm did not find the best state is that mlrose implements crossover somewhat randomly.

C. *Flip Flop*

The flip flop problem evaluates a current state vector *x* of bit string values where maximum fitness is obtained by having each consecutive value different. In other words the values flip flop (Mlrose.readthedocs.io, 2019):

$$x_i \neq x_{i+1} \quad \forall\, i \in x$$

This problem was evaluated as a discrete optimization problem. It has two global maxima, 01010… and 10101… .

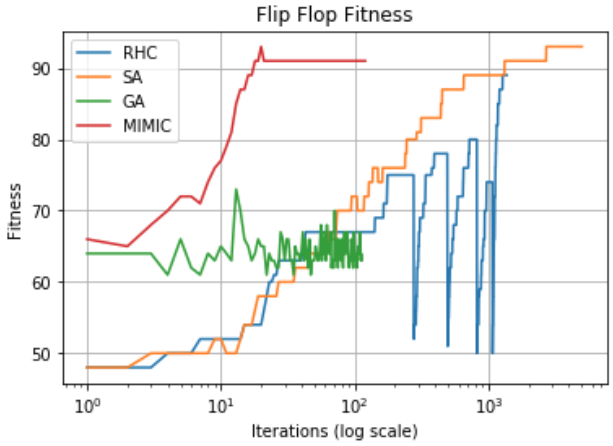

Fig. 12. Flip flop comparison of all four algorithms fitness vs. iterations (log scale)

I optimized the flip flop problem in mlrose with $n = 100$. I ran all 4 algorithms over 5,000 iterations. I implemented early stopping if fitness was not improved over 100 iterations. The optimal fitness is 100.

Table III shows a summary of the statistics for each search method. For random hill climbing I optimized the number of restarts. The algorithm converged to a fitness of 89 with 4 restarts after approximately 1,250 iterations.

For simulated annealing I optimized the decay schedule of the temperature parameter. The algorithm converged to a fitness of 93 with an exponential decay of the temperature after approximately 2,700 iterations.

For the genetic algorithm I optimized the population size and mutation rate. The algorithm did not find the optimal state. With a population size of 250 and a mutation rate of 0.14 the algorithm's best fitness was 73 after approximately 11 iterations.

TABLE IV. STATISTICS OF LOCAL RANDOM SEARCH OF ONE MAX FITNESS FUNCTION

Model	Best Fitness	Iterations	Search Time (sec.)
Random Hill Climbing	89	1250	0.1572
Simulated Annealing	93	2700	0.6655
Genetic Algorithm	73	11	10.3074
MIMIC	93	20	648.5845

On the MIMIC algorithm I optimized the population size and the percentage to keep. The algorithm converged to a fitness of 93 with a population size of 300 and a 35% keep percentage after 20 iterations.

Fig 10 shows the comparison of each algorithm's fitness over the number of iterations. I assumed MIMIC would have the largest fitness since it is able to retain the structure from previous states and flip flop is based on its alternating structure. I was surprised at how well random hill climbing and simulated annealing were able to do. Figs. 11 and 12 show that for larger n the fitness for these two methods would be around 80-85%.

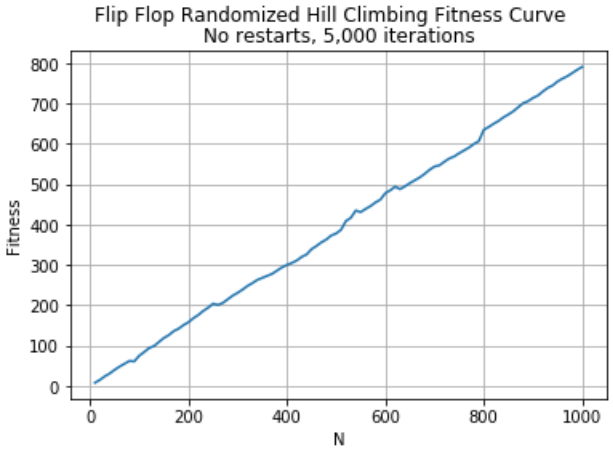

Fig. 10. Flip flop randomized hill climbing fitness vs bit string length

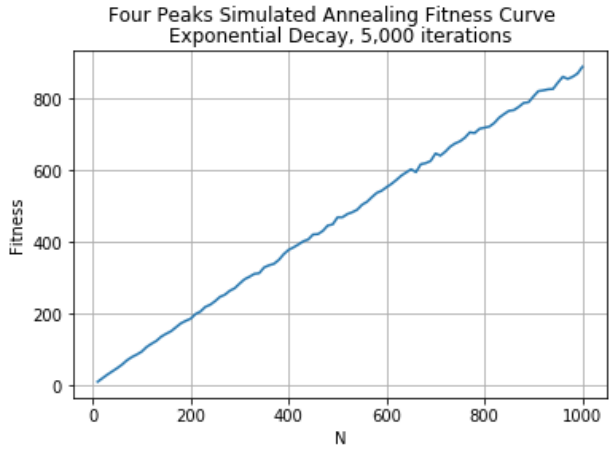

Fig. 11. Flip flop simulated annealing fitness vs bit string length

While MIMIC would continue to achieve a fitness higher than 90%.

IV. CONCLUSION

It is clear to see that each algorithm has their own strengths and weaknesses. It is clear that domain knowledge is important when selecting your search method.

Random hill climbing finds a better neighbor that leads to a local optima that isn't necessarily a global optima. While there are certain techniques to help avoid getting stuck at these, they are not guaranteed to find the best solution.

Simulated annealing jumps around the domain much more to avoid getting stuck at these local optima before becoming more specific in their search and therefore can perform better.

While the genetic algorithm does well in decision spaces that are not very complex. It does not hold on to the structure of the space. Therefore, it makes jumps in fitness and then keeps search for a better fitness to jump to. This works well when there is locality in your domain. However, it can get stuck at suboptimal fitness spaces while it searches for a better one.

MIMIC is very powerful and does well on most problems by taking into account the previous states and structure in its search. However, its power is not always necessary when you factor in the computation time of the various search methods.

V. References

[1] Mlrose.readthedocs.io. (2019). *mlrose: Machine Learning, Randomized Optimization and SEarch — mlrose 1.0.0 documentation*. [online] Available at: https://mlrose.readthedocs.io/en/stable/index.html [Accessed 4 Mar. 2019].

[2] Baluja, S. and Davies, S. (2019). Using Optimal Dependency-Trees for Combinatorial Optimization: Learning the Structure of the Search Space. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.433.6088&rep=rep1&type=pdf [Accessed 4 Mar. 2019].