Abstract

This analysis comprises of two sections, with the first comparing three randomized search algorithms to backpropagation with gradient descent for tuning a neural network and the second comparing four randomized search algorithms applied to three different common optimization problems. The mlrose[1] library is heavily utilized throughout along with some optimizations[2] made to the mlrose library.

Randomized Search Algorithms

The four randomized search algorithms we will be comparing in this analysis are Random Hill Climbing, Simulated Annealing, a Genetic Algorithm, and MIMIC.

Random Hill Climbing is an optimization algorithm that starts at a random X-value and climbs towards the local optimum on each iteration. Once a local optimum is reached, the algorithm can restart at another randomly chosen X-value if restarts are allowed. This alteration to the standard hill climbing algorithm can offer significant improvement at a constant factor increase in complexity. In this experiment random hill climbing was not given any restarts so we can compare how simulated annealing modifies this algorithm to improve performance.

Simulated Annealing is a modification on the hill climbing algorithm that uses search and exploration to find an optimum. The algorithm goes through an iteration by sampling a point and then selecting the next sample based on an acceptance probability also known as the temperature. A high temperature will make the algorithm behave more randomly, and as the temperature approaches zero, the algorithm behaves more like a hill climbing algorithm. The temperature is decreased on each iteration based on a decay algorithm, of which we will experiment with Arithmetic Decay and Exponential Decay. The benefit of this method is it is less likely to get stuck at a local minimum than hill climbing. The name is based on a metallurgy analogy in which repeated heating and cooling strengthens the blade.

Genetic Algorithm models biology by using the notion of populations, mutations, crossovers, and generations. A new generation is created from a population where the most fit individuals are kept, and the least fit individuals are replaced via crossover. A generation in our case is an iteration. Mutation refers to the random randomized local search within a population. Crossover refers to the combining of parents to form a child in the next generation.

MIMIC is a randomized search algorithm that uses dependency trees to maintain history and structure and uses this knowledge to search using less iterations than the other algorithms. The algorithm iterates as follows:

1. Generate random population of candidates from uniform distribution, median fitness is drawn from this data as θ_0
2. Update density estimator params

[1] https://github.com/gkhayes/mlrose
[2] https://github.com/parkds/mlrose

3. Generate more samples from the distribution
4. Set θ_{i+1} to Nth percentile of data, retain points < θ_{i+1}

Finding Optimal Neural Network Weights

We will start by comparing the performance of the first 3 randomized search algorithms to backpropagation with gradient descent for estimating the best weights of a neural network. The neural network will only use 1 hidden layer with 10 nodes since those parameters were found to be the best in terms of speed compared to performance on the chosen dataset from assignment 1. The dataset is a binary classification on credit card default. Using factors related to payment history, sex, gender, and age, the neural network tries to predict whether the individual will default on the next months payment.

Overall, backpropagation performed with the highest accuracy, but took the second longest to train the model. One could argue that random hill climbing performed the best overall since it was only 3% less accurate but 88% faster than backpropagation. The genetic algorithm performed the worst overall. It gave less than 1% higher accuracy than random hill climbing but took 2 orders of magnitude more time to complete the optimization. Simulated annealing performed the exact same as random hill climbing, with a slightly higher clock speed. The temperature parameter was not high enough to make the algorithm perform differently than random hill climbing. The poor performance of the genetic algorithm can be attributed to the continuous search space of neural network weights.

Algorithm	Train Time (sec)	Query Time (sec)	Training Accuracy	Testing Accuracy	Iterations
Backprop with Gradient Descent	5.1579	0.0120	0.8067	0.8202	100
Random Hill Climbing	2.7371	0.0103	0.7757	0.7913	100
Simulated Annealing	3.9999	0.0099	0.7757	0.7913	100
Genetic Algorithm	486.8882	0.0130	0.7870	0.7982	100
Backprop with Gradient Descent	49.7490	0.0100	0.8167	0.8322	1000
Random Hill Climbing	32.2152	0.0189	0.7757	0.7913	1000
Simulated Annealing	51.8499	0.0105	0.7757	0.7913	1000
Genetic Algorithm	1534.3313	0.0120	0.7870	0.7982	1000

Figure 1. Training and testing timing vs accuracy for various neural net tuning methods.

Optimization Problem Performance Comparison

Next, we are going to solve three different optimization problems using the four randomized search algorithms. The optimization problems include the traveling salesman, continuous peaks, and four peaks.

Traveling Salesman

The first optimization problem we are solving is the Traveling Salesman problem. The problem takes in a set of coordinates, and outputs a vector of integers representing the optimal order in

which to visit the coordinates. For this problem, optimal is defined as the minimum Euclidean distance to visit all the points. To optimize the problem and avoid visiting the same point more than once, we are using the mlrose TSPOpt optimization class. Figure 2 shows the grid that was generated for this problem. The grid consists of 100 randomly generated points on a 100x100 grid. It is interesting because as the problem scales, it becomes NP hard to solve. All four algorithms perform very differently as the number of points increases.

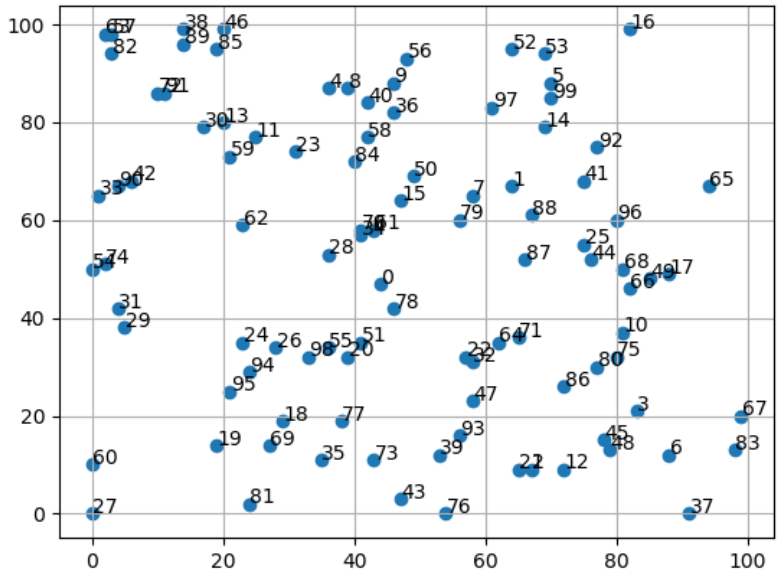

Figure 2. Plot of traveling salesman points between which to minimize distance.

Initially the experiment was run with the default parameters for each of the randomized search algorithms and the max iterations parameter was tuned. Each algorithm could run up to the max number of iterations but didn't need to run through all the iterations if it converged earlier. Figure 3 shows the fitness and timing performance between the four algorithms with the fitness being the minimum distance between the points. In this experiment, a lower fitness score indicates better performance. Genetic Algorithm and MIMIC both converged around 10 iterations while Random Hill Climbing and Simulated Annealing took 100 iterations to show a significant performance increase. Only 100 points and 100 iterations were initially used due to the exponential time and space increase for the MIMIC algorithm as iterations and points are added. MIMIC shows great performance in terms of fitness and can converge on 10 iterations but due to MIMIC needing to build a dependency tree to maintain structure, it doesn't scale as well to larger datasets on traveling salesman. This problem has $0.5 * (n-1)^2$ possible solutions, so going from 100 to 1000 coordinate sets is 2 orders of magnitude higher. The way in which this problem is designed makes the search space much broader than the continuous peaks and four peaks problems.

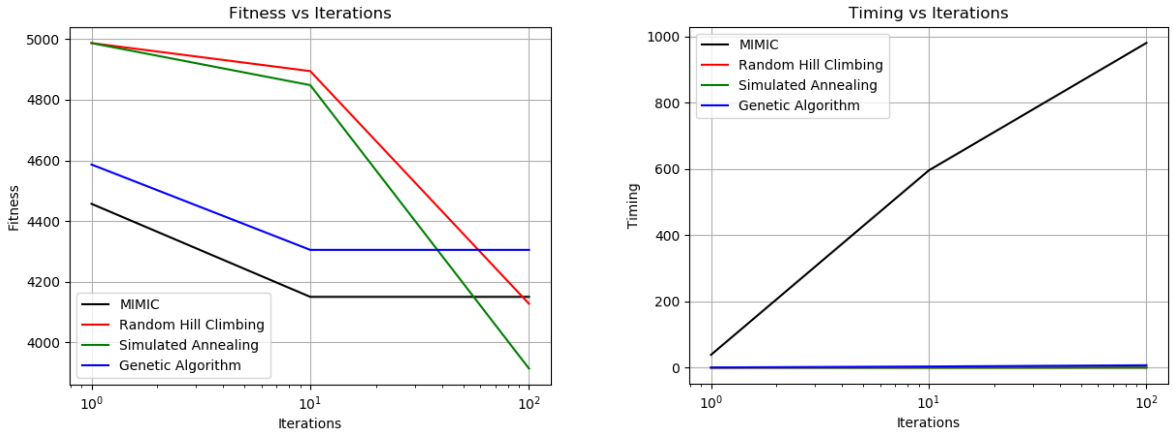

Figure 3. Timing and fitness curves for traveling salesman optimization problem.

The nature of the genetic algorithm makes it seem like a good candidate for the traveling salesman due to its early convergence and consideration of points residing in a "population" on a 2D plane. To better test the genetic algorithm performance, a new experiment was designed to visit 1000 points in a 100x100 grid. The results are summarized in Figure 4. The genetic algorithm converged again on 10 iterations and was able to outperform random hill climbing and simulated annealing. Both random hill climbing and simulated annealing rely on a lot of iterations as the search space grows so they can perform improve via restarts. Simulated annealing outperforms random hill climbing due to its temperature parameter which allows it to jump more when the temperature is high.

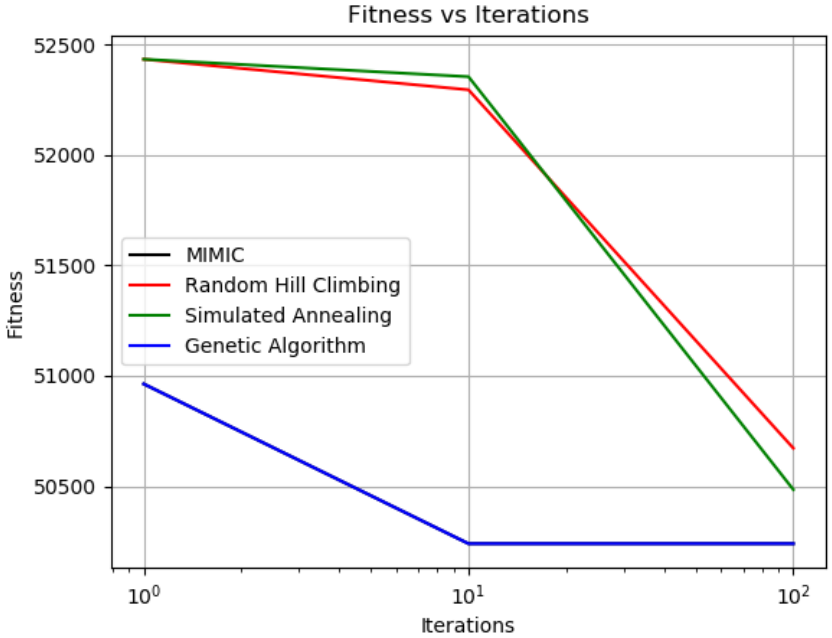

Figure 4. Fitness curve of traveling salesman using 1000 points, not including MIMIC due to memory constraints of the program runtime.

Given the performance of the genetic algorithm in the pervious experiment, another experiment was created to test different population and mutation rates. Figure 5 summarizes the results, with the first part testing different population sizes and the second part taking the "best" population size and testing different mutation rates. Having a larger population in the genetic does not necessarily increase fitness and significantly increases the time taken to converge the algorithm. This is due to the algorithm needing to evaluate more candidates at each iteration. The table shows that 200 was an ideal population size, it seems that a lower population did not include enough candidates and a higher population took too much time per iteration. In general, the lower mutation rate performed better than a higher mutation rate. The lower mutation rate allows the algorithm to be more selective, which works well on the traveling salesman problem due to the large number of data points. Mutation rate did not have a significant impact on the time complexity.

Population	Mutation Rate	Fitness	Time	Iterations
50	0.20	4369.5195	1.6072	10
100	0.20	4459.5397	3.3098	10
150	0.20	4385.9448	3.1474	10
200	0.20	4304.8817	4.0993	10
250	0.20	4395.9602	5.2304	10
300	0.20	4474.9138	5.3279	10
500	0.20	4366.2965	8.5973	10
1000	0.20	4382.9404	16.5393	10
2000	0.20	4307.8553	36.6701	10
200	0.10	4304.8817	4.2114	10
200	0.30	4447.2257	4.0476	10
200	0.40	4494.1859	3.9621	10
200	0.50	4407.3062	5.8914	10
200	0.75	4502.3209	4.9016	10
200	1.00	4429.9503	4.0973	10

Figure 5. Table comparing mutation and population size for genetic algorithm on traveling salesman optimization.

Continuous Peaks

Continuous peaks is an optimization problem defined as the max of the longest run of 0's or 1's plus a value *n* if the max run exceeds the temperature parameter or more formally:

$$Fitness(x,T) = max(\, max\_run(0,x), max\_run(1,x)\,) + R(x,T), \textit{ where,}$$

$$max\_run(b,x) = length\ of\ the\ max\ run\ of\ b\text{'s}\ in\ x, \textit{ and}$$

$$R(x,T) = \begin{cases} n, if\ (\max\ \_run(0,x) > T\ and\ \max\ \_run(1,x) > T) \\ 0 \end{cases}$$

This is an interesting problem since there are many local optima. The input to the problem is a random bit string of length 1000. Given the large search space it could be easy to get stuck at a local optimum. Simulated annealing performs the best on this problem due to the temperature parameter with exponential decay allowing for random jumps early on. In looking at the results in Figure 6, we can see it took the simulated annealing algorithm 10000 iterations in order to achieve finding the best maximum. MIMIC also showed good performance with a convergence at 10 iterations and the best fitness at that point.

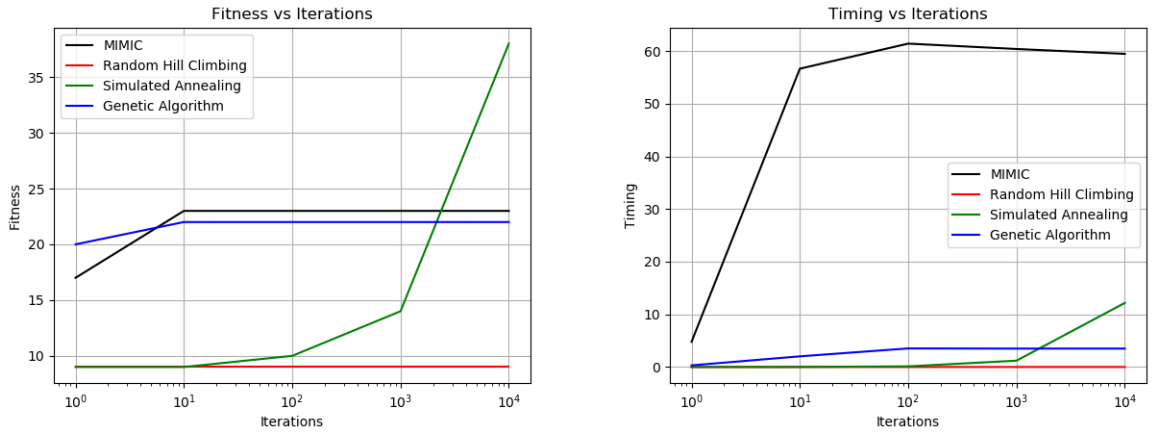

Figure 6. Timing and fitness curves for traveling continuous peaks optimization problem.

The simulated annealing algorithm showed the best performance by far in terms of fitness and performed well with regards to time. We are going to take a further look at the decay algorithms used along with the temperature. A further experiment was conducted to compare exponential decay to arithmetic decay along with different starting temperatures. The experiment kept the best parameters from the initial experiment with a max iteration rate of 10000 iterations. The decay rate for arithmetic decay was increased as the temperature increased so that the temperature could decay proportionally. The decay rate for exponential decay was left constant since the exponential factor decreases at an exponential rate. Figure 7 shows the results of exponential decay and arithmetic decay with starting temperatures between 0.1 and 100. The exponential decay showed the best fitness score as the temperature increased. This is due to the nature of simulated annealing; with a high temperature the algorithm will make more random jumps in the early stages and slow down exponentially as the temperature cools or decays and behave like a traditional hill climbing algorithm. This makes the algorithm less likely to get stuck at a local optimum.

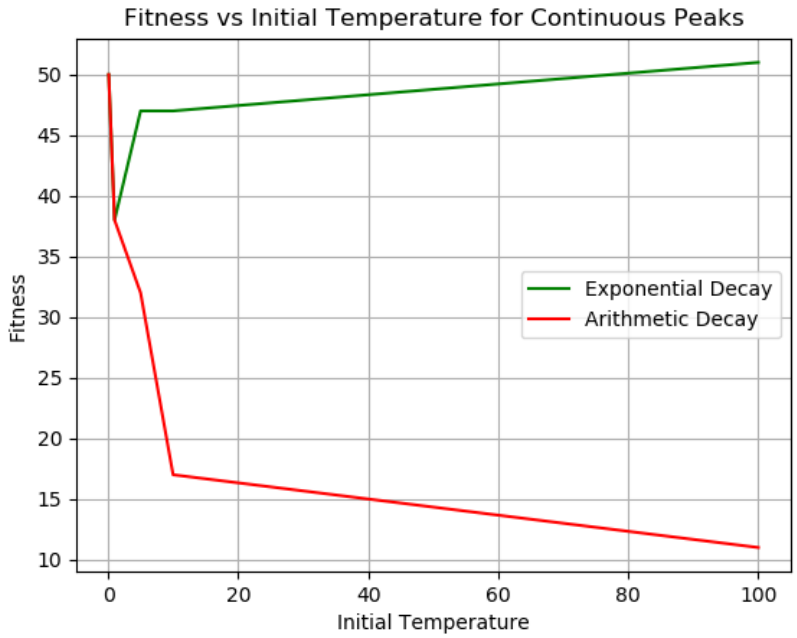

Figure 7. Comparison of exponential and arithmetic decay based on initial temperature.

Four Peaks

The four peaks optimization is defined by,

$$Fitness(x,T) = max(\, tail\,(0,x), head\,(1,x)\,) + R(x,T), \text{ where}$$

$$tail\,(b,x) = number\ of\ trailing\ b\text{'s in } x$$
$$head\,(b,x) = number\ of\ leading\ b\text{'s in } x, and$$

$$R(x,T) = \begin{cases} n, if\ (tail\,(0,x) > T\ and\ head\,(1,x) > T) \\ 0, otherwise \end{cases}$$

This is an interesting problem as it contains two optimum global maxima and two local maxima that are less than the optimum. The global maxima occur at points where there are $T+1$ leading 1's followed by all 0's or where there are $T+1$ trailing 0's preceded by all 1's. The difficulty of this problem increases exponentially in difficulty as the value of T increases.

I would argue that the genetic algorithm and MIMC have the best performance on this optimization as they both converge around 10 iterations and simulated annealing does not return any local optimum until it hits 10000 iterations.

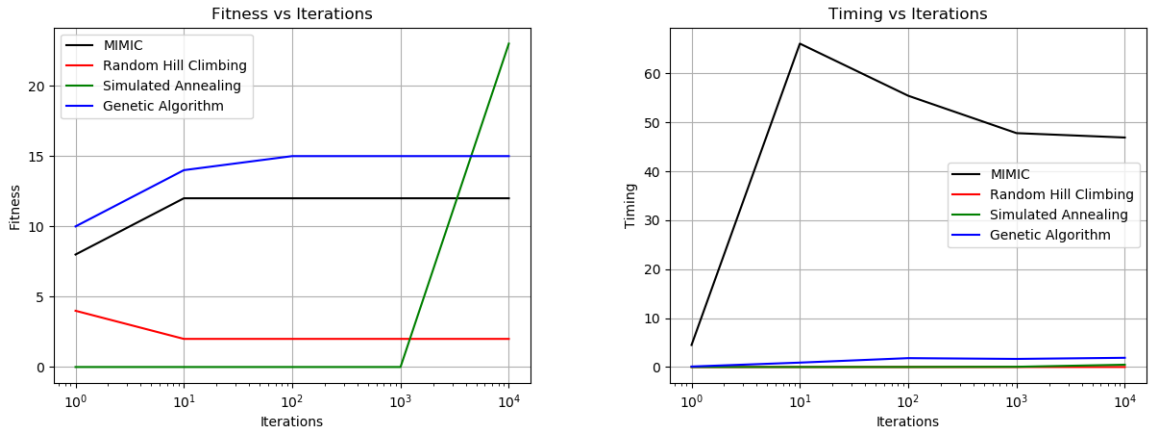

Figure 8. Timing and fitness curves for four peaks optimization problem.

Conclusions

Given the results of the four algorithms performing across the different problems they all have strengths and weaknesses. Random hill climbing tends to operate the fastest, but also tends to underperform as it frequently gets stuck at local optima. We did not give the random hill climbing and restarts, so we could compare it closely to simulated annealing and how the temperature parameter can offer performance improvement.

Simulated annealing showed great performance on the continuous peaks problem especially when the temperature was adjusted. As the initial temperature increased, it allowed the algorithm to make more jumps early on and eventually settle on a more optimum maximum.

The genetic algorithm performed best on the traveling salesman problem with the larger map since it was able to use few iterations of mutations to converge, and it does not have to store the structure like MIMIC does.

MIMIC converges with less iterations, would be good for problems where evaluation at each iteration is costly. However, as shown in the traveling salesman problem, when the search space is very large, MIMIC can struggle due to the structure it wants to generate which can exceed memory constraints. This could be due to the implementation in mlrose.