Randomized Optimization

Daniel Bach
dbach7@gatech.edu

1 INTRODUCTION

In this analysis, several local random search algorithms will be analyzed in order to discover their trade offs and benefits. To do this, four algorithms will be used against three different discrete optimization problems. The algorithms explored are: randomized hill climbing, simulated annealing, a genetic algorithm, and MIMIC. The optimization problems testing these algorithms will be: the One-Max problem, the Flip-Flop problem, and the Four-Peaks problem. In the second part of this analysis, the algorithms will be used in place of backpropagation to find the weights of a neural network model.

2 DISCRETE OPTIMIZATION PROBLEMS

In this section, the four algorithms will attempt to maximize the fitness functions of the three discrete optimization problems. The two measures used in comparing these algorithms will be the wall clock time for the algorithm to run as the problem size increases and the fitness score achieved. These two measurements will be averaged across ten runs for each problem and problem size.

The wall clock time is an important measurement as it is a good heuristic of how computationally expensive an algorithm will be. This is important as the algorithm ought to be practical in application so that solutions can be provided as soon as possible [1].

The goal of all three optimization problems is to maximize the fitness function, so it is important to see the fitness score achieved by these algorithms. It will also be interesting to see if different algorithms will outperform another at different problem sizes.

My initial hypothesis for all problems regarding wall clock time is that randomized hill climbing will be the fastest as the algorithm is very cheap in that it guesses some x and moves in the direction of the neighbor of a higher fitness score. The next fastest algorithm should be simulated annealing as simulated annealing also has a similar process in that the algorithm jumps to a new point given a probability function and the temperature. In regards to genetic algorithms and MIMIC, it is unclear without looking at the results which would be faster as they both have more computationally expensive calculations. Genetic algorithms create a population of sample inputs

and then perform crossovers from the more fit individuals. MIMIC also creates a population of sample inputs from a probability distribution and only keeps samples above a fitness threshold. The genetic algorithm could take slightly more time as it must also perform the crossover step.

All of the algorithms and fitness functions used for this analysis were implemented in the mlrose_hiive library which can be viewed at [2]. For the following three subsections (2.1, 2.2, and 2.3). Parameters for the optimization algorithms which are not the default values are listed below:

random_hill_climb [3]: max_attempts=50, restarts=10

simulated_annealing [3]: max_attempts=50

genetic_alg [3]: max_attempts=50

mimic [3]: max_attempts=50

2.1 One-Max problem

The One-Max problem is a simple bit-string optimization problem where the fitness function [4] is given by:

$$Fitness(x) = \sum_{i=0}^{n} x_i$$

Meaning that the bit-string of length n can have a maximum fitness score of n, when the bit-string is all 1's. For example, the bit-string 01101 would have a score of 3 while the maximum score achievable is 5 (with bit-string 11111). The goal of this problem is to maximize the fitness function to achieve a score of n.

My initial hypothesis for this problem is that all algorithms should be able to do well as there is only one global maxima for each problem size, which is the bit-string consisting of all 1's. Since all algorithms do well, the differentiating factor would be wall clock time. With this criteria, randomized hill climbing should be the 'best' performing algorithm in this problem.

2.1.1 Results

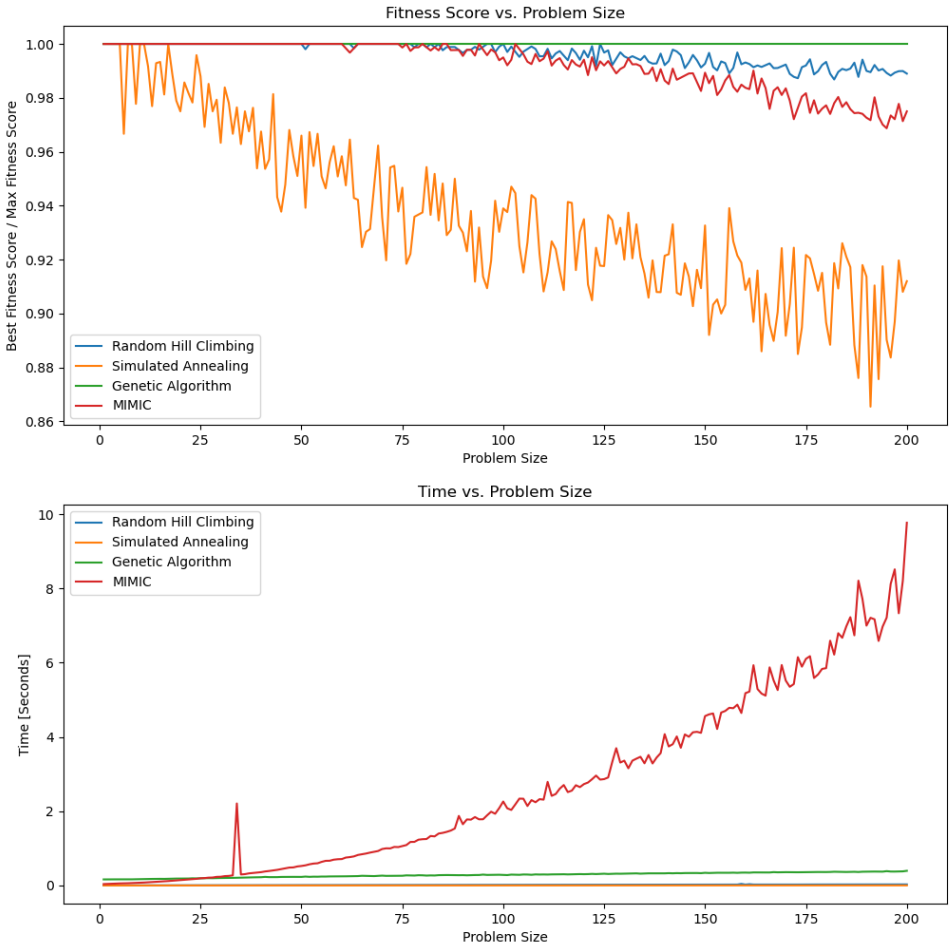

Figure 1—Results of the One-Max problem

The Y-axis on the fitness score vs. problem size chart is the average best fitness score over 10 runs / the maximum fitness score as it was easy to calculate the maximum fitness score of the One-Max problem (max score = n). All algorithms performed well at relatively small values of n from approximate n=1 to n=10.

After n=10, simulated annealing performed worse than all three other algorithms, with the gap in performance growing with the problem size. The minimum temperature for the simulated

annealing problem is 0.001, as T goes to 0, the results should become more similar to random hill climbing but that is not seen here. The genetic algorithm finds the global optima for every problem size from n =1 to n=200. Random hill climbing performed well as expected since there is only one global maxima and no local optima that random hill climbing might get stuck in. I would expect that random hill climbing would also be able to find the max fitness score if allowed more iterations and restarts.

As expected, simulated annealing and randomized hill climbing did not take very long to finish iterating. The genetic algorithm did not take very long as well. These three algorithms seem to almost not take much longer as the problem size increases. However, MIMIC has an exponential relationship with the problem size while still not performing as well as the genetic algorithm or randomized hill climbing.

Taking wall clock time and performance into account, I would say the genetic algorithm performed the best as it was able to always find the global maxima, even when averaged across ten runs for each problem size. It did not take much longer than randomized hill climbing either.

2.2 Flip-Flop problem

The Flip-Flop problem is a bit-string optimization problem where the fitness score is determined by the total number of consecutive pairs which are different-valued [4].

$$Fitness(x) = \sum_{i=0}^{n-1} 1, \text{ if } x_i \neq x_{i+1} \text{ else } 0$$

To maximize the score of this fitness function, the bit-string should be a repeated bit-string of alternating bits (e.g., 101010 or 01010) of length n.

My initial hypothesis for this problem is similar to the One-Max problem since there is only one global maxima. All four algorithms should be able to perform well on this problem so the differentiating factor would be time, unless there is a big difference in fitness scores across problem sizes.

2.2.1 Results

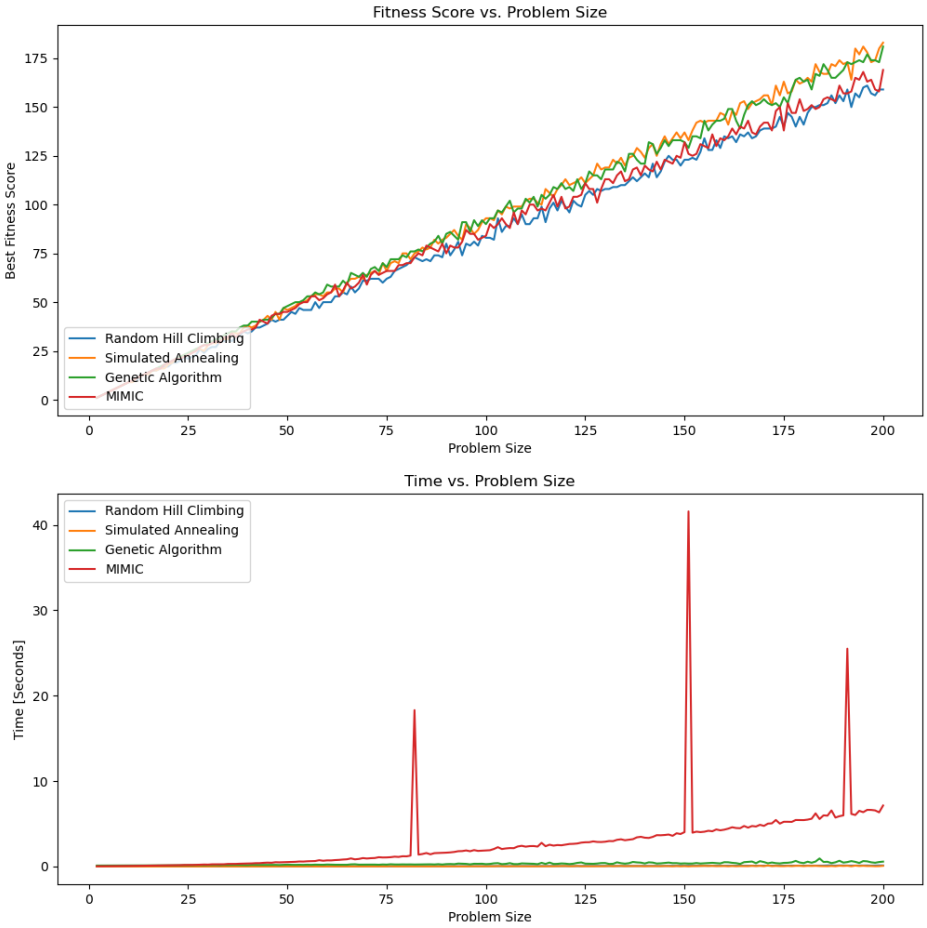

Figure 2—Results of the Flip-Flop problem

Looking at the fitness score chart, it's not apparent that any one algorithm clearly out performed another one. At n=200, the genetic algorithm and simulated annealing achieved similar results, scoring a little higher than MIMIC and randomized hill climbing. Since the genetic algorithm does crossover with other fit individuals from the population sample of inputs it generates, it makes sense that it is able to perform well in the Flip-Flop problem. It would be able to create a more fit individual by taking sequences of alternating bits from the fit individuals.

Looking at the wall clock time chart, MIMIC displays an exponential relationship with the problem size as it also did with the One-Max problem. Interestingly, there are a few input sizes where the MIMIC algorithm takes a very long time to run. This could be due to outliers in the runs as it was run on my M1 Macbook Air and there are other processes occurring simultaneously with the code running.

Overall, I would say the simulated annealing performed the best in the Flip-Flop problem as it achieved similar fitness scores to the genetic algorithm while taking slightly less wall clock time.

2.3 Four-Peaks problem

The Four-Peaks problem is a bit-string optimization problem where the fitness score is given by [4]:

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

Where:

$tail(0, x)$ is the number of trailing 0's at the end of the bit-string

$head(1, x)$ is the number of leading 1's at the end of the bit-string

$R(x, T)$ is a bonus term adding to the fitness score which equals n if $tail(0, x)$ and $head(1, x)$ are both over a threshold $T = t_{pct}n$. For this analysis, $t_{pct} = 0.10$.

My initial hypothesis for this problem is that randomized hill climbing will not perform as well since there are four maximas with two of them being smaller, local maximas. The two smaller maximas occur when the bit-string is all 0's or 1's as that creates a fitness score of n for a bit-string of length n but not including the $R(x, T)$ bonus term. To reach either of the two global maximums, the bit-string must maximize the length of the tail or head term while reaching the threshold on both ends as well to get the bonus term added to the fitness score.

2.3.1 Results

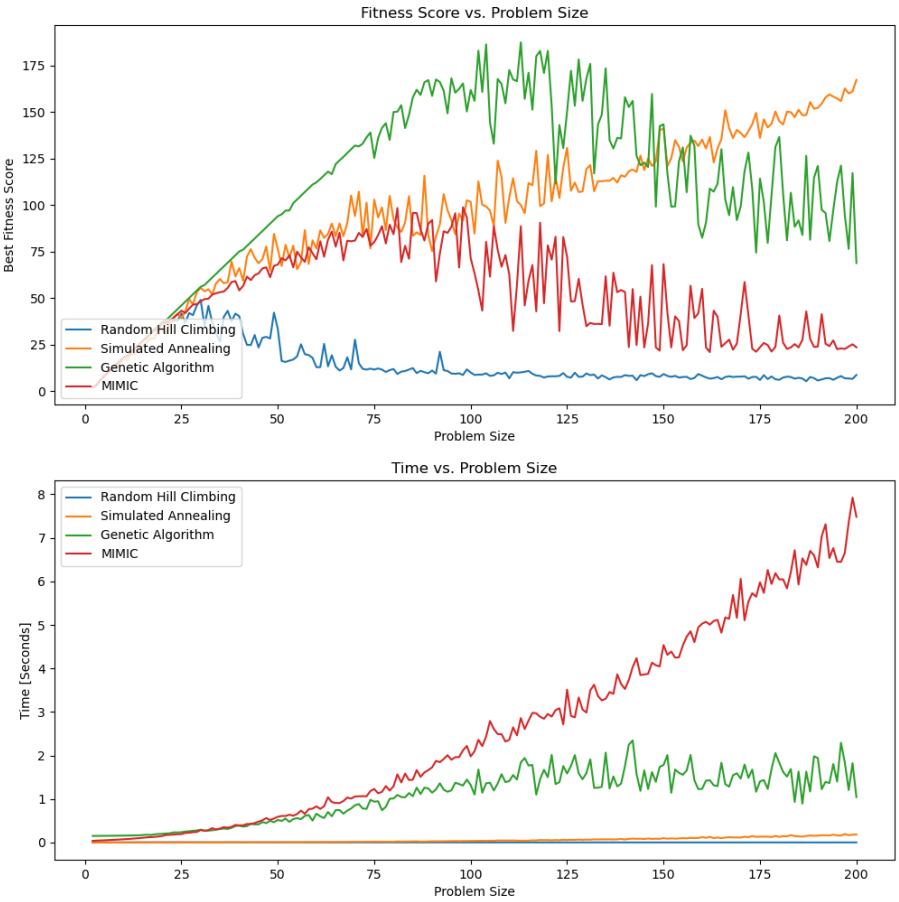

Figure 3—Results of the Four-Peaks problem

Looking at the fitness plot, all algorithms achieved similar performance for n=2 to n=25. After this point, their performance varies greatly. From n = 25 to n=150, the genetic algorithm outperforms the three other algorithms. For n=150 to n=200, simulated annealing performs the best. Randomized hill climbing performs the worst for every n. This can be explained by the fact that this problem has two local maximas which are smaller than the two global maximums. The randomized hill climbing algorithm might be converging onto the local maximas. Simulated annealing performs the best here, maybe due to the fact that it is a combination of exploitation

and exploration. It is able to explore areas closer to the global maximums and areas outside of the two smaller local maximas.

Interestingly, the genetic algorithm seemed to grow with the problem size here whereas it did not with the One-Max and Flip-Flop problem. Simulated annealing and randomized hill climbing still took very little time while MIMIC displayed an exponential relationship with the problem size.

3 OPTIMIZATION ALGORITHMS APPLIED TO NEURAL NETWORKS

In this section, the three random optimization algorithms will be used in place of backpropagation to update the weights of a neural network model. The dataset used will be the Iris dataset which was loaded using scikit-learn [5]. The data contains 150 instances with three possible labels. Features are real-valued and continuous measurements. Labels are {0, 1, 2} which represents the type of iris the instance is.

In regards to preprocessing, the features are scaled using scikit-learn's MinMaxScaler [6] which scales each feature from 0 to 1. To use mlrose_hiive's neural network model , the labels must be one-hot encoded and transformed into a vector [7] since it is a multi-classification problem.

The three random optimization algorithms will be compared to a MLPClassifier [8] neural network model that was used previously in this class. The MLPClassifier model has the following parameters: alpha=1e-4, max_iter=1000, hidden_layer_sizes=(50,50), random_state=8, learning_rate='constant', and early_stopping=True.

For the neural network model using randomized hill climbing [10], the parameters are: restarts=10, max_iters=1000, hidden_nodes=(50,50), early_stopping=True, and random_state=8.

For the neural network model using simulated annealing [10], the parameters are: max_iters=1000, hidden_nodes=(50,50), early_stopping=True, and random_state=8.

For the neural network model using a genetic algorithm [10], the parameters are: rmax_iters=1000, hidden_nodes=(50,50), early_stopping=True, and random_state=8.

3.1 Learning Curves

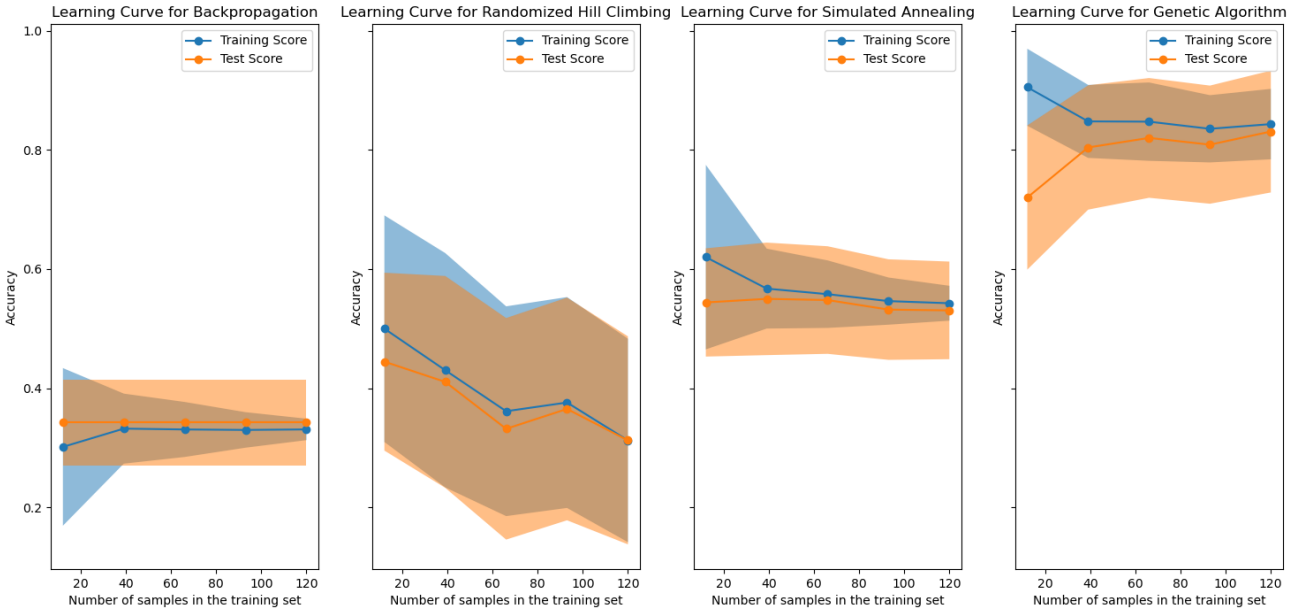

Figure 4—Learning curves for each method of updating the neural network weights

These learning curves were generated by varying the amount of training data the models had access to, then testing the model on its training data and a test set. The bands around each line represent the uncertainty of the model which was determined by cross-validation [10]. Across all models, those that performed better on the training data were also able to perform nearly as well on the test set. Interestingly, the model using backpropagation performed the same on the test set regardless of how many training samples it had. It was expected that randomized hill climbing would not be a good choice of a weight update algorithm for the neural network model since the weights are continuous values and could result in many local optima for the algorithm to get potentially stuck in. Simulated annealing performed better than randomized hill climbing and this could be due to the fact that it was able to explore more choices of the weight combinations before going into exploitation-mode and converging onto a set of weights. Lastly, the genetic algorithm performed the best across all the algorithms, achieving the highest accuracy scores on the training and testing set. It also clearly was able to predict better as more training data became available to the model. This makes sense as there are many nodes in the hidden layers and the genetic algorithm would be able to create combinations of weights from the fittest individuals of the population to make a stronger model.

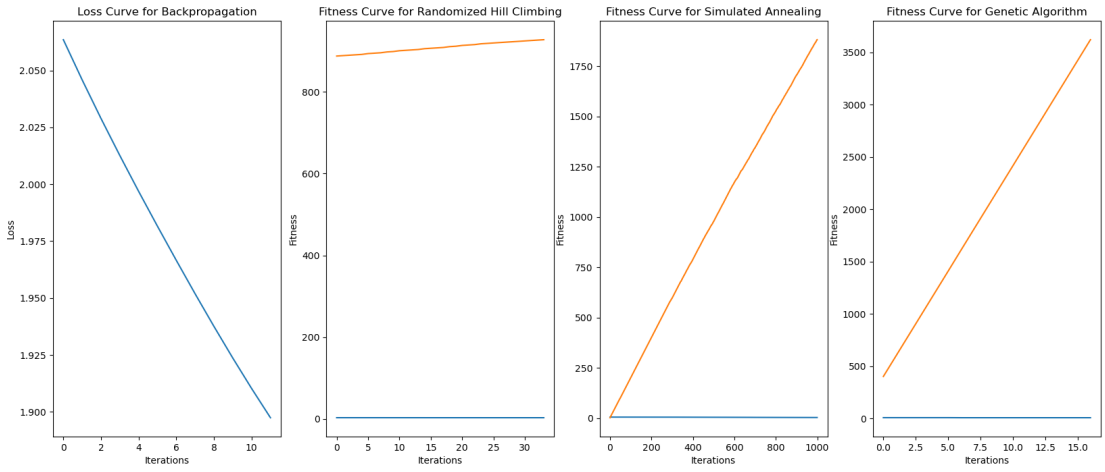

Figure 5—Loss curves for backpropagation, fitness curves for the three random optimization algorithms. Ignore blue lines for the fitness curve plots.

Examining the loss and fitness curves, the result that stands out is that the randomized hill climbing algorithm does not perform much better with more iterations. Like before, this can be explained by the fact that there might be many local maximas of the fitness function that the algorithm is able to get stuck in. The other three algorithms clearly improve and minimize loss / maximize the fitness score as they iterate.

4 REFERENCES

1. https://chat.openai.com/share/85729aec-c041-47c8-ac18-4be05b04c463
2. https://github.com/hiive/mlrose
3. https://mlrose.readthedocs.io/en/stable/source/algorithms.html
4. https://mlrose.readthedocs.io/en/stable/source/fitness.html
5. https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html
6. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
7. https://mlrose.readthedocs.io/en/stable/source/tutorial3.html
8. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
9. https://mlrose.readthedocs.io/en/stable/source/neural.html
10. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.LearningCurveDisplay.html