Randomized Optimization

Exploration of Randomized Optimization algorithms

Roshan Gajurel

Georgia Institute of Technology

Rgajurel3@gatech.edu

Abstract—This paper explores four local random search algorithms and makes comparison among them.

I. Introduction

This paper focuses on four of the most often used Randomized Optimization techniques:

- Randomized Hill Climbing
- Simulated Annealing
- Genetic Algorithm
- MIMIC

We will analyze the algorithms using two different methods and explore it in different sections below. In the first section, we will use Neural Networks to analyze these algorithms. We will use Random Hill Climbing, Simulated Annealing and Genetic Algorithm as the optimization algorithm and compare it to back propagation used by neural networks. We will measure how these algorithms will find the weights on a same dataset.

In the second section, we will compare Simulated Annealing, Genetic Algorithm and MIMIC algorithms by analyzing their performance in three different maximization problems. Since, we want to compare the strengths and weakness of each of the algorithms, I have chosen problems such that each of these algorithms will do better than other in at least on of these problems. The problems that I have chosen are Traveling Salesman, 4-peaks and Max k-Color.

I used mlrose libraries exclusively to run all the analysis.

II. Randomized Optimization in Neural Network

In this first part of the paper, our goal is to find the best weight for a Neural Network for the Wisconsin dataset which was used in my first paper on Supervised Learning as well. I have primarily used iteration, learning rate and maximum attempts in each point as common hyperparameters to compare these algorithms. Other specific hyperparameters are tuned for specific algorithms.

A. Back Propagation

Neural networks in general use backpropagation which uses gradient descent as the optimizing algorithm. It can learn non-linear models in real time using partial fit. However, since the hidden layers can have more than one local minimum, random weight initialization can lead to different validation accuracy.

Let us look at the Error curve against the learning rate for this algorithm. Learning rate at any point is the rate at which the algorithm will look for the next point in its neighborhood. If the point is too big, it might overshoot and miss the optimal point. If the rate is too small, it will be more accurate to find the optimal point but will be slow.

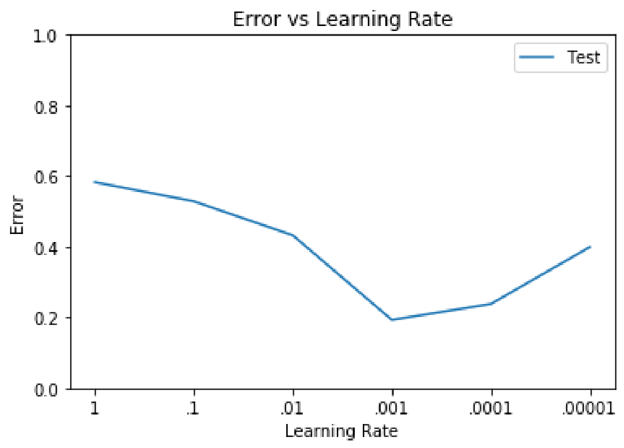


Fig. 1 Gradient Descent Error vs Learning Rate

We can see that as the error decreases as the learning rate decreases. We can see that .001 is the optimal learning rate and gives the optimal weight with the error of around .2.

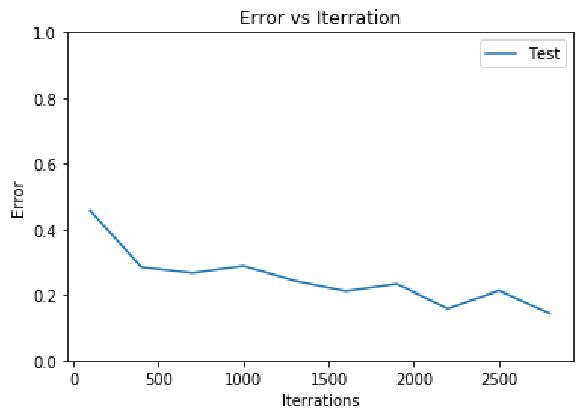


Fig. 2 Gradient Descent Error vs Iteration

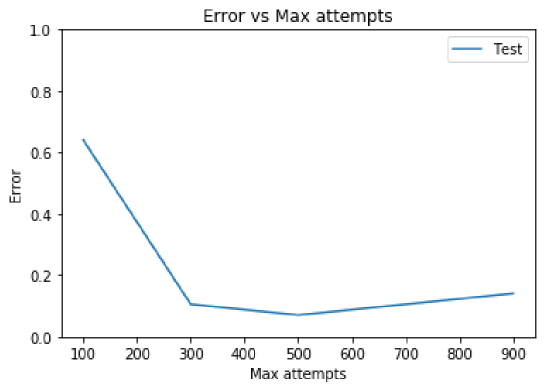


Fig. 3. Gradient Descent Error and max attempts

Next, we use Iteration and Max attempts as the other two hyperparameters to find the best weight. Looking the graphs, we can see that the error goes down as the iteration increases which makes sense because the algorithm gets more chances to find the global optima and not get stuck in a local optimum. In terms of wall time gradient descent took 3 seconds per iteration in average.

By tuning the hyperparameters, I was able to get a best accuracy score of 97.3% on a never before seen data at which point the algorithm would have the best weight this will be the baseline for these comparisons.

B. Randomized hill climbing

Randomized hill climb tries to randomly find a starting place for a function and finds the optimal point from the starting place. Therefore, the algorithm might just find the local optima and not the global optima if it does not get lucky when it chooses the start point.

We will use the same three hyperparameters to find the best weights for randomized hill climbing as well, which are

Iteration, learning rate and max attempts.

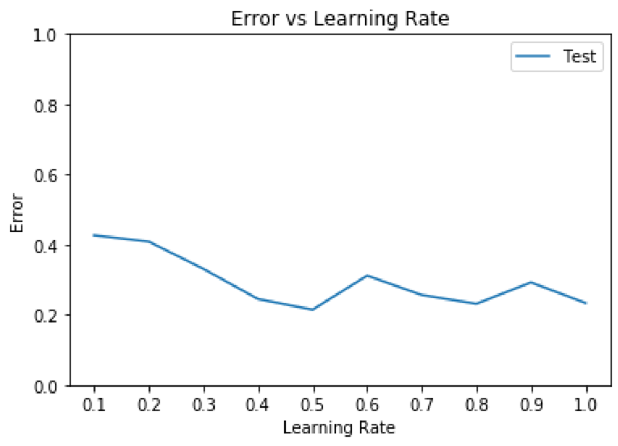
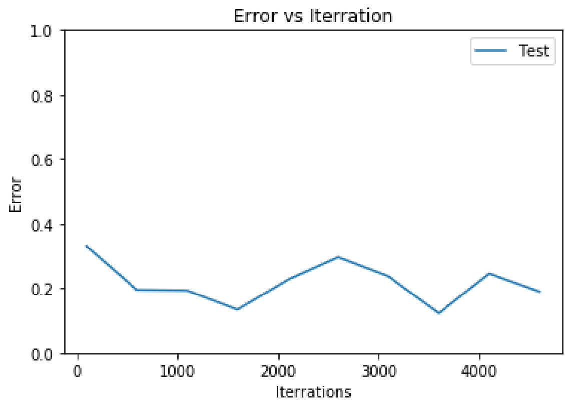


Fig. 4. RHC Error vs Learning Rate

First we look at the effect of learning rate on the error. In the graph we can see that error is least between 1 and .1. In the above graph we see that learning rate is the best around .5. So we can consider .5 to be the point were it does not over shoot and miss the optimal solution.



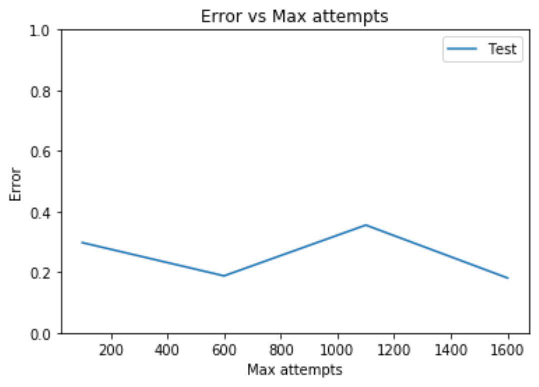


Fig. 5 RHC Error vs Iteration and max attempts

Looking at the error and iteration curve we can see that the algorithm does well around 1500 attempts and does not seem to do any better after that. Also, the best max attempt seems to be around 600 after which it does not do significantly better. This is because, when using a lower number to look for the optimal point in the neighborhood it might not be able to find it because of the limited number of available attempts. However, after 600 attempts it is able to confidently find the optimal point. In terms of wall time RHC took 2.03 seconds per iteration in average.

Using the best hyperparameters to get the most optimal weights, I was able to get a best accuracy score of 97.1% with random hill climbing. However, RHC can achieve this with far less iteration as well as a lot higher learning rate compared to backpropagation even though backpropagation had better accuracy.

C. Simulated Annealing:

Simulated Annealing is like Random Hill Climbing in that it chooses different starting point each time to converge to find an optimal point however instead of choosing the starting point at random it uses an acceptance probability function to choose the next point. It uses temperature to decide the probability of the next starting point. However, we should be careful when changing the temperature. It should be decreased slowly to avoid from being stuck on a local minimum.

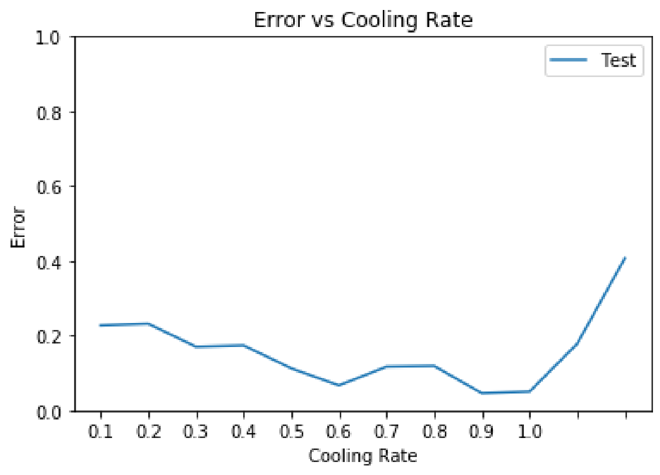
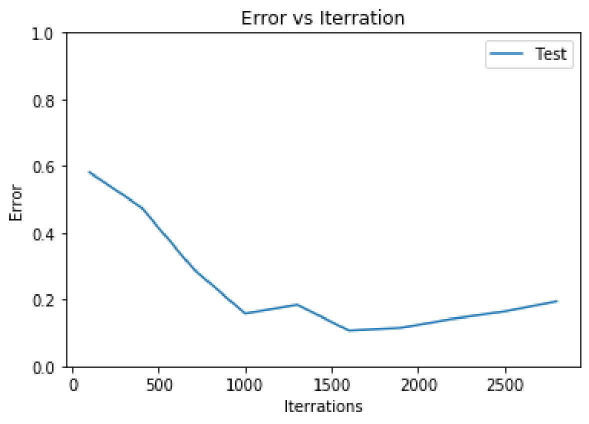


Fig. 6 SA Error vs Learning Rate

As before we look at the Learning Rate, the error is the lowest when the learning rate is between .1 and 1, as we can see that .9 is the lowest point.



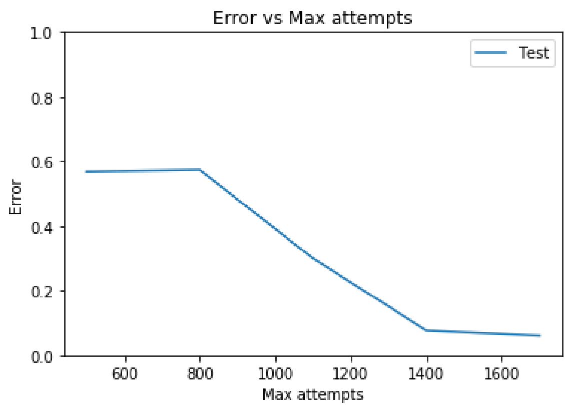


Fig. 7 SA Error vs Iteration and max-attempts

Observing the graph for iteration we can see that it does best around 1500 iteration and does not perform any better after that. This is because it can find the optimal starting in those 1500 iterations. The graph for Max attempt is very clear. There is a very high correlation between the number of attempts and the error. For this data set it seems like as the number of attempts increases from 800 to 1400, the accuracy increases significantly. Also, I used geometric decay as the decay schedule which did better than arithmetic decay and Exponential Decay for this dataset. In terms of wall time SA took 2.9 seconds per iteration in average.

Using the best hyperparameters, the neural network got the best accuracy of 97.3%. At this point the it has the best weight.

D. Genetic Algorithm

Genetic Algorithm takes inspiration from natural selection and evolution. Genetic Algorithm start by picking a population from the dataset and then it finds the fitness of the population. It picks the top most fit data and then it uses mutation or crossover to replace the least fit population. This process continues until it converges, these iterations are called generations. GA increases the diversity of the population which helps in it having better chance of getting to the global optima rather than being stuck local optimum.

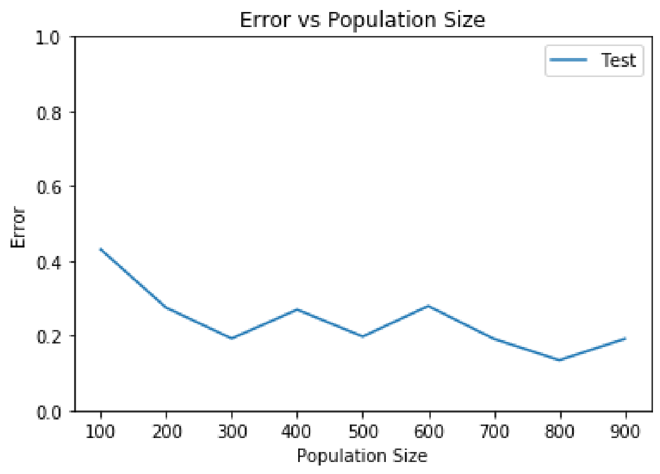


Fig. 8 GA Error vs Pop Size

For Genetic Algorithms, mutation probability and population size are the prominent hyperparameters. In the first graph, error against population we can see that the error decreases as the population increases. It gets to the lowest error around 800. This is probably because as the population increases it has a larger data set to choose the most fit data and perform mutation and crossover.

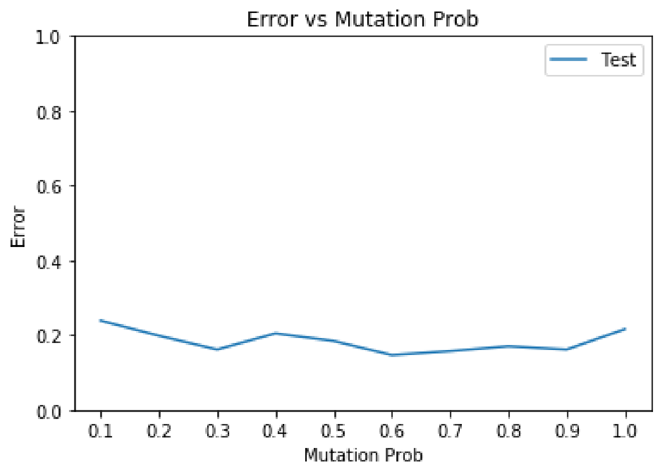


Fig. 9 GA Error vs Mutation Probability

Also, we can see that mutation probability is best around .03 but it does not affect the error in a lot, this is probably because the size of the data is small. In terms of wall time GA took 1.28 seconds per iteration in average.

Using the best hyperparameter the optimized network using Genetic Algorithm gives a best test accuracy of 95.6% which is not as good as the other algorithms.

Conclusion:

After comparing all the algorithms and tuning their hyper parameters we can conclude that back propagation using gradient descent gives the best accuracy. Random Hill climb and Simulated Annealing can do better if the number of iterations is increased. At one point, it is sure to find the global optimum. However, GA did not do as well as other algorithms. It is probably because the size of the data used. Other algorithms were able to perform better on a smaller data size. However, in terms of wall time GA performed the best with a time of 1.28 seconds per iteration.

III. Optimization Problems

According to Russel and Norvig (2010)[1] the aim of an optimization problem is to find the best state according to an objective function. In this case we are trying to find the best parameters which will maximize the function. We have chosen three interesting problems for investigation. They are, Four Peaks, Max-k-color and Khapsack. We will use these problems to highlight the advantages of one optimization algorithm over other. We will use for optimization algorithms namely, Genetic Algorithm, Simulated Annealing, MIMIC, Random Hill Climb for comparison and will use the first three for comparison.

A. Four Peaks:

Four peaks is a optimization problem in which can be defined as, given N-dimensional input vector X and parameter *T,*

F(X, T) = max [tail(0, X), head(1,X)] + R(X, T)

Where:

tail(b, X) = number of trailing b's in X,

head(b,X) = number of leading b's in X

R(X, T) = N if tail(0,X)> T and head (1,X) >T; and

R(X, T) = 0 otherwise

This function has two global maxima and also two sub-optimal local maxima that occur with a string of 1's or 0's.[2] The complexity of this problem increases with the increase in the value of T since the basin of attraction for the inferior local maxima becomes larger. However, for this assignment we will be maximizing for the function.

For this problem T was .10 of N and the size of N was 30.

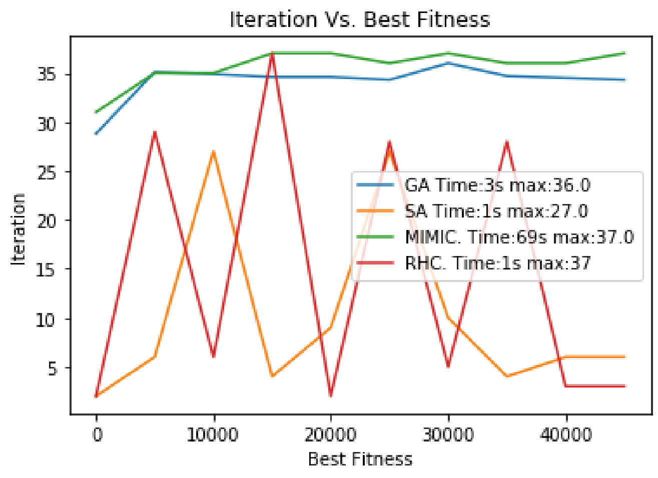


Fig. 10 4 Peaks Iteration Vs Fitness

MIMIC does a lot better than other algorithms by far. It can find the global maximum for this problem in just around 500 iteration. Other algorithms seem to converge to other sub-optimal local optima. However, MIMIC takes a lot longer than other algorithms per iteration. It is interesting to note that RHC eventually finds the global maxima and GA comes very close to it as well but never gets there. It can also be noted that GA is probably the second-best algorithm for the problem. Both MIMIC and GA do better on this problem because by working on a subset of the problem and iterating over it they are able to reach the optimal solution more consistently. MIMIC does better because it is able to deliberately determine the structure while iterating in contrast to GA which has to build knowledge generationally. In contrast to RHC and SA where they try a random point every time it, does not do find the global optima every time.

B. Max k-color

Max k-color problem can be best understood as a map coloring problem. For each node it assigns one color K such that no adjacent nodes have the same color. Or trying to find the least number of colors to color a map of countries so that no adjacent countries have the same color. However, we are trying to maximize the fitness function so the outcome will maximize the number of adjacent pairs that are colored the same. This is a NP-Complete problem.

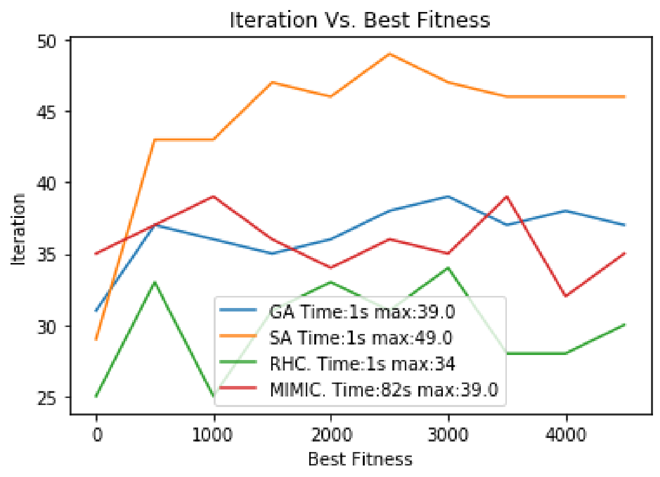


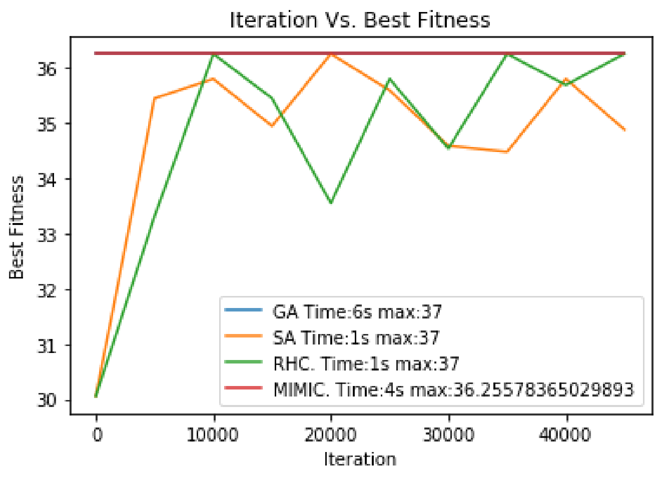
Fig. 11 Max k-Color Iter. Vs Fitness

For this problem I used 50 as the number of nodes. We can see that SA performs the best in this case by a fair margin. Mimic and GA do almost the same but MIMIC takes a lot longer. RHC does the worst. Since SA uses probabilities ot find the next starting point, this problem is suitable because using probability estimation it can find the parameter to maximize the function faster than other algorithms.

C. Travelling Salesman Problem:

The scenario for a travelling salesman problem can be described in a situation where a travelling salesman needs to travel to n cities and needs to find the most optimal path to travel to all cities and return back to the original location. However, since we are maximizing this problem, we are finding the longest path to all the cities.

I have 2 graphs for this problem the first one is for eight cities, and the second is for 10 cities, which is more complex.



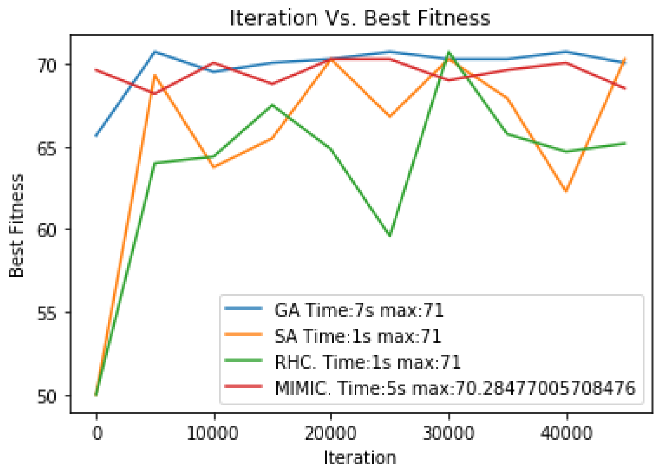


Fig. 12 TSP Iteration. Vs Fitness

We can see that on a simpler problem with 8 coordinates bot GA and MIMIC do similar, actually GA finds the maximum solution 37 consistently where as MIMIC is stuck around 36.25. When we increase the complexity of the problem a little, we can see that GA performs better with the best overall fitness. We can see that RHC and SA also find the maximum, but it is very random and lucky. GA is able to do it consistently. This is because GA uses the smart technique of learning over generations and mutating to find the optimal solution. Since it is a NP hard problem, finding by luck is hard so the nondeterministic approach GA takes is better compared to other algorithms in this case and is able to outperform other algorithms.

IV. Conclusion

We explored four different randomized optimization algorithms using two different scenarios. From the first scenario we can conclude that Gradient Descent for back propagation was the better of the algorithms for the dataset that I had. In terms of wall time RHC and SA are faster, and GA and MIMIC are slower.

We also analyzed the strengths and weaknesses of these algorithms using three different problems. We found out that MIMIC does better in 4 peaks, SA does better in Max k-map and GA does better in Travelling salesman problem. Even though some of these problems were NP Complete, these algorithms were able to solve them with relative ease. We can conclude GA and MIMIC give consistently better results as well as are suited to solve more complex problems. RHC and SA are faster, and SA can perform better in situation were probabilities can give an advantage.

References

[1] Russell, S. and P. Norvig (2010). *Artificial Intelligence: A Modern Approach*, 3rd edition. Prentice Hall, New Jersey, USA..

[2] De Bonet, Jeremy S., Charles Lee Isbell Jr, and Paul A. Viola. "MIMIC: Finding optima by estimating probability densities." *Advances in neural information processing systems*. 1997.