Randomized Optimization Report

Lijun Huang  
[lhuang349@gatech.edu](mailto:lhuang349@gatech.edu)

# Abstract

This report will conduct experiments on three different discrete optimization problems, namely one-max, four-peak and traveling salesperson, by using different randomized optimization algorithms, including Randomized hill climbing (RHC), Simulated Annealing (SA), genetic algorithm (GA) and MIMIC. In addition, there are also a comparison performed with first three algorithms to the backprop neural network model that was calibrated in assignment 1. The mlrose [1] package and sklearn [2] was used for the implementation of the algorithm mentioned above.

# Three Optimization problem

## One Max –- Best performance algorithm: GA

### Problem description

The one max optimization problem has the following fitness function, it has an n-dimensional vector as input state. Where each x can take either 0 or 1.

This problem is a bit string discrete-valued parameter spaces problem. Intuitively, this would be very easy for human to solve. However, it might not be that intuitive for algorithms as it cannot comprehensive the problem the same way as we do.

In addition, this optimization problem can highlight the advantages of the genetic algorithm. The reason is that we can optimize each of the component individually. As this is a simple summation, maximize each input vector will maximize the fitness function. One of the fundamental assumptions for genetic algorithm to work is that the problem can be optimized by each component individually, and we can get better offspring by using cross over between good parents.

### Result analysis

In the figure 1 below shows a sample fitness learning curve by using those 4 algorithms. Y axis is the fitness value, x axis is number of iterations to converge, and we are trying the maximize the fitness function. The experiments were run with the following parameters: the max attempt was set to 100 for all four algorithms, the population size for GA and MIMIC are set to 1000, mutation probability for GA is 0,05, the keep percentage for each iteration in MIMIC is set to 0.2.

Chart

Description automatically generated

1. Learning curve for one max problem with vector length 20

From the above figure, we can see that GA takes much less iteration to converge than RHC and SA. However, we can see MIMIC took less iteration than GA. This is not surprising, as MIMIC is almost the same as GA, except is also keeping the structure for the probability distribution, whereas GA only keeps points.

Looking at number of iterations will not tell us the full story, in the figure 2 below, we can see that the time it took for MIMIC growth exponentially, whereas all other three seems not taking much longer time to converge as the complexity of the problem growth.

Chart, line chart

Description automatically generated

1. Highest fitness value and convergence time respect to different problem length

As the complexity growth, the maximum convergence fitness value for GA is not performing as well as SA and MIMIC. I think the reason is that mlrose is doing random permutation when generating the offspring, with the complexity of the problem growth longer and longer, the number of permutations increase exponentially, it’s easier to stuck into the local maxima and not able to find optimal solution. However, when the problem complexity is low, GA can perform very well, and the nature of the fitness function is consistent with the fundamental assumption of the GA algorithm.

## Four Peak – Best performance algorithm: SA

The four peak problem has a fitness function as follows:

Where:

* , is the number of trailing and leading b’s in x, respectively.
* = n if both and > T, = 0 otherwise

The parameter setting was the same as the one max problem for those algorithms. This problem is bit sting optimization problem. In this problem, complexity is defined as the input vector length, the possible input value for each vector is 0 and 1. From the below figure, we can see that even though MIMIC can occasionally beat SA when the complexity is high, but MIMIC has exponential running time compare to other algorithms and the performance is not stable. On the contrary, GA’s performance is much more stable and higher compare to other algorithms.

Chart, line chart

Description automatically generated

1. Highest fitness value and convergence time respect to different problem length

The reason should be that GA is very useful in situation where there are many local optima, whereas algorithm like RHC can easily be stuck at local maxima. This can be confirmed by the fact that RHC’s performance is always very low in the above graph. The decay schedule used in SA is exponentially decay schedule, this would allow SA to explore more possibility at the beginning stage and less randomization in the later process to help achieving better performance and potentially more likely to find the global maximum.

## TSP – Best performance algorithm: MIMIC

Traveling salesman problem is NP hard problem, the problem can be formulated as this: give a list of cities and distance between each pair of cities, finding the shortest possible path that visit each city just once and return to the original city. In order to formulate the fitness function to maximize it, the negative distance was returned, this ensures the largest fitness value is our desired results.

Chart, line chart

Description automatically generated

1. Highest fitness value and convergence time respect to different problem length

We can see before the complexity reaches 60, that is, total number of cities below 60 (the number of edges were generated randomly during the process), MIMIC performs reliably better compare to other algorithms. However, 4 algorithms began to have similar performance as complexity continues increase, this might be due to the fact that the problem goes too complex, no algorithms can find the global maxima results. However, one of the problems we still need to be aware with MIMIC is that the exponential time it took to converge as the complexity increase.

The reason why MIMIC performs better in this NP hard problem should be due to the fact that it can analyze the structure of the optimization problem, it can “communicate” cost function and probability distribution obtained to the next iteration to improve the performance. Whereas other three algorithms only have information about points. In addition, MIMIC attempts to discover common structure about the optimal by computing second-order statistics and do sampling from a distribution consist with it.

# Find good Neural network weights

## Introduction

This section will try to use three randomized optimization algorithms to determine the optimal weights for a neural network and compare to the backprop neural network used in the assignment 1. This comparison was performed by using the bank marketing dataset [3] that were used in the last assignment (21 attributes with 4119 observations in total, which related to a marketing campaign of a Portuguese banking institution)

In order to improve model performance, different parameters are adjusted, like population size, mutation probability, max attempts and learning rate. But all seems not working well in term of performance improvement. In the below section will cover learning rate as example to try to explore this.

To ensure proper comparison, randomized optimization algorithm used same setting as the backprop one. With 5 layer of network and 300 nodes at each layer, using relu as the activation function.

## Result analysis

Model results for learning rate = 0.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| model | training\_accuracy | testing\_accuracy | f1\_score | roc\_score | training\_time |
| random\_hill\_climb | 0.1378 | 0.1311 | 0.1919 | 0.4986 | 1.8596 |
| simulated\_annealing | 0.1378 | 0.1408 | 0.1955 | 0.4982 | 34.1609 |
| genetic\_alg | 0.9008 | 0.8920 | 0.2764 | 0.5810 | 681.8867 |
| backprop | 0.9733 | 0.8835 | 0.5200 | 0.8910 | 26.9685 |

Model results for learning rate = 0.01

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| model | training\_accuracy | testing\_accuracy | f1\_score | roc\_score | training\_time |
| random\_hill\_climb | 0.1384 | 0.1286 | 0.1859 | 0.4874 | 1.9046 |
| simulated\_annealing | 0.1357 | 0.1408 | 0.1973 | 0.5031 | 38.7683 |
| genetic\_alg | 0.9032 | 0.8665 | 0.1538 | 0.5413 | 696.7049 |
| backprop | 0.9706 | 0.8811 | 0.4368 | 0.8812 | 31.0609 |

Model results for learning rate = 0.005

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| model | training\_accuracy | testing\_accuracy | f1\_score | roc\_score | training\_time |
| random\_hill\_climb | 0.1420 | 0.1141 | 0.1667 | 0.5072 | 3.4918 |
| simulated\_annealing | 0.1357 | 0.1408 | 0.1973 | 0.5031 | 50.1182 |
| genetic\_alg | 0.9032 | 0.8665 | 0.1538 | 0.5413 | 711.0930 |
| backprop | 0.9706 | 0.8811 | 0.4368 | 0.8812 | 20.1727 |

As first, this result was puzzling me, seems I’m not able to get a better result no matter what parameters I tried. The best those algorithms can achieve is just slightly better than random by looking at the ROC score. Adjusting parameter seems not have much meaningful impact on the results. For example, in the above, three different learning rates are experimented. The ROC score for the randomized algorithms all seems to close to be at around 0.5, and the accuracy score for RHC and SA are just terrible.

One of the hypotheses I thought is due to the imbalanced nature of the problem. In the ban k marketing dataset, only 451 out of 4119 observation are “yes”, that consist less than 10% of the data. This might cause the randomized algorithm to be stuck at local maxima and not able to explore the full picture, since RHC and SA are both some sort of greedy algorithm, it will stop once we are no longer able to improve the performance, this might be able to explain the poor performance of them.

One another hand, GA can achieve somewhat high accuracy on this training and testing set, but still suffer in f1 score and roc score, which is still much worse than the backprop NN. The relative strength of GA compared to RHC and SA might be due to the property of GA, because GA can generate a population and can deal with more complex problem like non-linear problem and non-convex problems.

However, the random sampling method in the GA imply that we need to define a good crossover and mutation operations to actually get good offspring combination. Mlrose package currently does not support choices or custom setting on this, this can be one of the next possible area to look into to try to get better performance.

# potential next step

Explore some other setting for GA like different cross over and mutation, and the selection criteria. The mlrose package only has random mutation and cross over, with possibility to change population size and mutation probability. Adjusting cross over type might be necessary for different problems if we want to improve more performance.

Try run the neural network optimization on a different dataset, a more balanced one dataset might help improve the performance of those randomized optimization algorithm. Maybe also try a dataset with less complexity, to see how those factors will affect the performance of the algorithm.

# summary

In this report, 3 different discrete problem was introduced and experimented with different randomized optimization algorithms. Each algorithm has their advantages and limitations; therefore, we need to be careful when deciding the which algorithms to apply to the problem. That is to say, it is important that we have some prior knowledge to the problem and the algorithm we are using in order to achieve better performance.

In addition, optimize weight for neural network seems to be a more challenging question for the randomized optimization algorithm, it might be due to the complexity of the problem and the imbalanced dataset.

# References

1. Hayes, G. (2019). mlrose: Machine Learning, Randomized Optimization and SEarch package for Python. <https://github.com/gkhayes/mlrose>
2. Fabian Pedregosa, et al. "Scikit-learn: Machine Learning in Python". Journal of Machine Learning Research 12. 85(2011): 2825-2830.
3. Bank Data set: [Moro et al., 2014] S. Moro, P. Cortez and P. Rita. A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems, In press, <http://dx.doi.org/10.1016/j.dss.2014.03.001>. Available at: [bib] <http://www3.dsi.uminho.pt/pcortez/bib/2014-dss.txt>

# source code

Below is the link to the source code and data used for this report, more details of running the code can be found in file README.txt

<https://github.com/lijunujil/Gatech_ML_Course/tree/main/SupervisedLearningReport>