# CS7641 A2: Randomized Optimization

Trung Pham
*tpham328@gatech.edu*
https://www.overleaf.com/read/pcrfcjfphgcj#217a17

## I. Introduction

In this report, we aim to compare the effectiveness of three randomized optimization algorithms (Randomized Hill Climbing/RHC. Simulated Annealing/SA, and Genetic Algorithms/GA) in solving two optimization problems: the Four Peak problem and K-Coloring problem. Both of these problems are prebuilt in mlrose-hiive library that are often used to study optimization. In addition, we will apply these algorithms to optimize weights of a neural network instead of using traditional backpropagation.

Optimization is an essential part of machine learning. In assignment 1, we have experienced with the difficulty of finding optimal hyperparameters. A model can outperformed another significantly with good hyperparameters. Traditional optimization process often struggle with finding the global optima, especially when dealing with complex and high-dimensional problems. Our search usually stuck in the local optima. We will also experienced the trade off of exploitation and exploration to find a solution to escape the local optima effectively.

Simulated Annealing and Genetic Algorithms seemed to always outperform Randomized Hill Climbing due to the ability to escape local optima effectively as well as exploring other potential solutions. We would confirm that through the experiments in two optimization problems as well as optimization weights of a neural network process. In addition, although we can use the optimization algorithm to update the weights of a neural network; backpropagation is still a prefer choice due to its effectiveness and efficiency.

## II. Hypothesis

For both Four Peak and K-Coloring problem, Genetic Algorithms are superior optimization solution due to its ability to effectively explore the search space and avoid local optima. Simulated Annealing is also a good solution that required less computational power and perform better than RHC. RHC is likely the least effective solution due to its tendency to get struck in local optima or require substantial computational power to restart and escape from the local optima.

Additionally, while RHC, SA, and GA can be used to optimize weights for a neural network, traditional backpropagation remains the preferred choice to update weights. This is due to its ability to efficiently compute gradients and update weights in a direction that minimized error. It is more efficient in a way that leading to faster convergence and better performance. The three optimization algorithms however offered alternative ways in training neural network.

## III. Experiments and Algorithms of Optimization Problems

In this section, we will describe the two problem that are used to performance test the three randomized optimization algorithms. Both of these problems are popular problems that are often used to benchmark test optimization algorithms and available pre-built in mlrose-hiive library.

### A. Four Peaks

The Four Peaks problem we chose involved finding a binary string of length 50 that maximizes an objective function.

$$f(x) = \max(A(x), N - A(x)) + R(x)$$

where A(x) is the number of consecutive 1s from the start of the string, and the number of consecutive 0s from the end of the string. R(x) is reward function. $A(x) \geq t$ and $N - A(x) \geq t$ where $t = 0.15 \times 50 = 7.5$.

### B. K-Coloring

K-color problem is task of assigning colors to the vertices of a graph. We chose a graph with 50 nodes, and each node has a maximum of 5 connections. There are only 3 colors that can be assigned. The goal is to ensure that no two adjacent nodes have the same color, and minimizing the number of conflicts.

### C. Algorithms

In RHC, the first solution is randomly selected from all solution or states. RHC is a simple local search algorithm that explore neighbor solution to find improvement. After it find local optima, the process is reset and initial starting point is randomly selected again. After a certain number of restart, the best local optima is selected as global solution. Depending on the problem, the restart numbers are fine-tuned to find a optimal solution.

SA introduces a probability acceptance criterion that allows acceptance of worse solution to escape the local optima. If evaluation of neighbor state show improvement, it is selected (exploitation). Otherwise, it might still be selected with a probability (exploration). The balance between exploration and exploitation is controlled by a temperature that cool down after each evaluation process. We used geometric decaying algorithm with decay rate of 99 percent and lowest temperature of 0.001. The use of geometric decay allow the module to prioritize exploring as much as possible at the initial training process.

GA is a population-based optimization process. It is inspired by natural selection theory. It evaluate a population of candidate solutions. Through selection, crossover and mutation process, it evolve the solution population until optimal solution is born from the population. It is one of

the best optimization process due to wide search space and not stuck in local optima.

The algorithms are stopped after 1,000 iterations or when total of attempts to find better solution reach 100 and cannot find better solution. In case of RHC, it only stops after last restart is finished. We will only show in the graphs the RHC performance based on the best restart. All algorithms are used to solve two problems and compare fitness performance against numbers of iteration, problem complexity and computational cost.

To measure performance, we used fitness score and both problem are normalized to maximize the fitness function. On a bad day, RHC can only find local optima because its starting position are randomly selected. Even with big number of restart time, if starting position does not fall within the global optima domain, RHC will not find it due to its inability to escape local optima. Even for SA and GA algorithms, our results show a vast range of best fitness. To counter this randomness in performance, we repeated our experiments in 20 random states, and measure the fitness based on average score of best fitness from each random state. This was only done for two optimization problems and not the neural network experiment.

Additionally, we also considered FEvals and training time. FEvals is the numbers of function that were evaluated. A higher number mean more function is being evaluated which required more computational power so a more efficient module will have less FEvals. Our aim is to maximize fitness and minimize FEvals. Module training time is also being analyzed to capture the actual computational cost since different function might take different time to evaluate.

## IV. Analysis of Optimization Problems

### A. Four Peaks

In this problem, we used a pre-defined fitness function by the hiive library. It is used to evaluate how well a given binary string performs in terms of the problem objective. Our goal is to maximize the fitness score of the string solution.

We started out with different values of hyperparameters for each optimization algorithms. For RHC, it is the different number of restarts. For SA, it is the different number of starting temperature, higher temperature allowed more exploration steps. For GA, we tested with different number of population size and mutation rates.

There are many ways to solve Four Peaks, but our final module used to solve it are RHC with 50 restarts, SA with initial temperature of 500, GA with population size of 50 and mutation rate of 0.2. There might be a better hyperparameters set but is not be possible to train given our limited computational power.

Figure 1 showed the performance of three algorithms. The RHC curve looked not too bad because it is based only on the best restart turn. There are 49 other restarts that failed before RHC found global optima. Regardless of using best restart, RHC still came up short when compared with SA and GA algorithm. One way to improve RHC performance is increasing the number of restart at the cost of computation. With restart equal 50, it already took 24

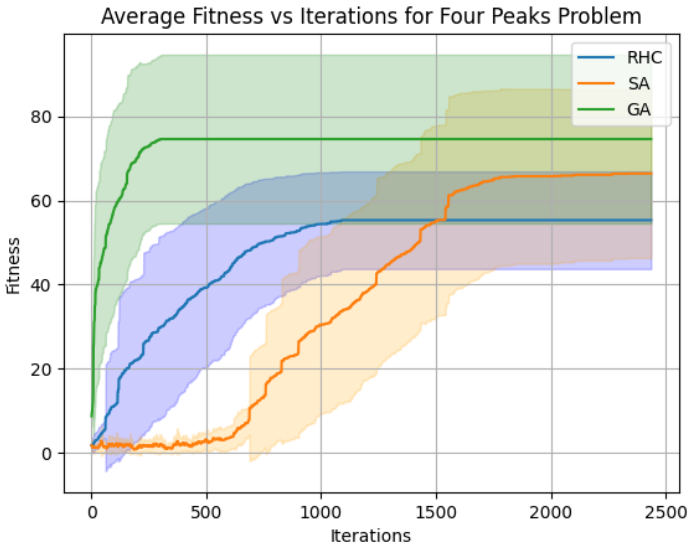


Fig. 1: Average best fitness for Four Peaks problem

seconds to train the module about 10 times longer than GA and 60 times longer than SA module. Keep increasing restart my not be a possible solution at some point.

When the random pointer fell within the global optima domain, RHC could find the solution quickly but this is by chance and we could not sample everything. RHC really struggle when the problem become more complicated.

| | Avg Best Fitness | Avg FEval | Training time |
|---|---|---|---|
| RHC | 55.3 | 23,954 | 24.2s |
| SA | 66.4 | 2,846 | 0.4s |
| GA | 74.6 | 15,690 | 1.9s |

TABLE I: Summary of performance metrics in solving Four Peaks

The next algorithm being analyzed is SA. We expected SA to be as good as GA and superior in term of efficiency. The exploration aspect of SA is controlled by a decay temperature. When the temperature cools down, SA gradually switch from exploration to exploitation. We picked geometric decay formula for the temperature because we would like to see SA explored more at the beginning and only prioritize exploitation near the end. In contrast with exponential decay where the temperature would cool down more quickly.

The average fitness score of SA in Four Peak problem is slightly smaller than GA's. It came as a bit of a surprise and part of the reason is because our decay hyperparameter might not be optimal. When solving Four Peak problem of shorter length or simpler problem, SA module successfully found global optima most of the time. It is as good as GA module in term of fitness score. SA can solve more complex problem but required more hyperparameters tuning.

When we looked at figure 2, performance vs. problem size. We can see that both SA and GA are on par with each other when dealing with more complex problem. RHC module will fell short quickly. Overall, both SA and GA module are excellent choice to solve Four Peak problem of any magnitude.

Additionally, The SA module is superior in term of efficiency. It required much less computational cost. In figure 3, SA evaluated least number of functions, followed by GA

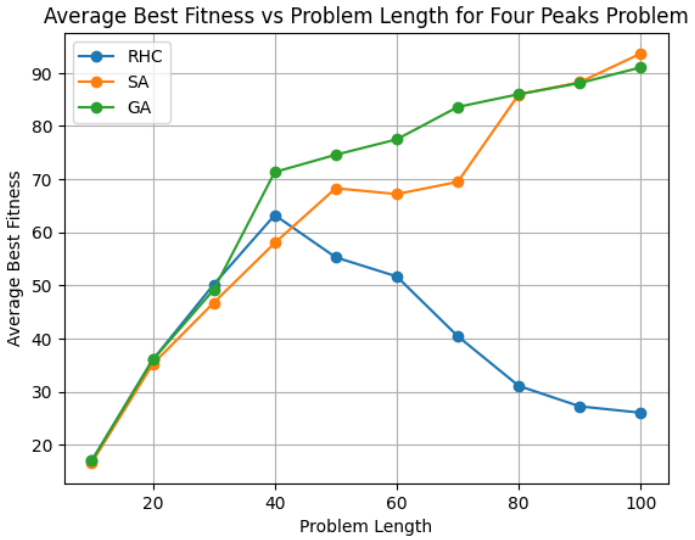


Fig. 2: Average fitness score for different problem size

and RHC. SA module evaluated 2,800 functions, about 5 times smaller than GA and 8 times smaller than RHC. Training SA module to solve Four Peaks only took 0.4 seconds compared to 1.9 seconds and 24.2 seconds of GA and RHC module respectively. SA algorithm is very efficient to solve. SA looked like it took longer iteration until it converged or solution is found. This is mainly because of the way to algorithm work that prioritize exploring at the beginning. Figure 1 did not capture the actual computational cost of RHC and GA module. RHC module looked like to converge after 1000 iteration but that is one of 50 repeated process, so we could not see the other 49 similar curves. GA module looked like to converge with least iteration but each iteration require evaluation, crossover and mutation process of the population of 50 states, in addition to the memory to store the population. In term of efficiency, SA module is truly superior.

SA module however require careful hyperparameter tuning. There is no one size fit all. For each problem length in Four Peaks, there are different optimal initial temperature and decay schedule to maximize fitness and minimize FEvals. Our experiments showed that geometric decay is more suitable for Four Peak problem than exponential or arithmetic decay due to the fact that SA module need more exploration to escape the local optima and Four Peak problem have quite a numbers of local optima. When we tuned the decay rate in geometric schedule, we realized that only initial temperature tuning is needed. Both decay rate and initial temperature are relatively correlated and have similar impact on the temperature under geometric schedule so we only need to tune one of them.

Finally, the GA module is also an excellent choice to solve for Four Peaks problem. GA module produced great fitness score, required a bit more computational power than the SA module and less than RHC module. GA module can solve Four Peak problem even when the problem get more complicated. GA also does not require much hyperparameter tuning as SA module as long as we found a good combination of population size and mutation rate. We found many optimal solution with different mutation rate below 20%. As population size increase, the GA module performance also increases at the cost of computational power. Therefore, we decide to use population size of 50 and mutation rate of 20% so the module converge faster with good result. We will discuss more about GA in K coloring problem next.

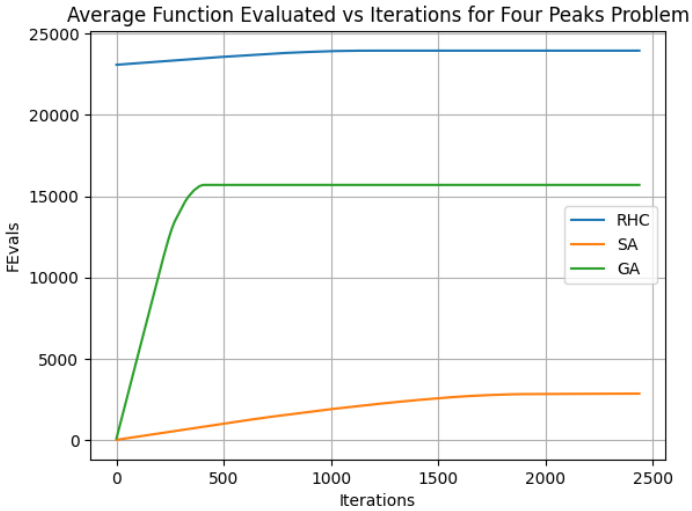


Fig. 3: Average Functions Evaluated for Four Peaks problem

### *B. K-Coloring*

In our chosen K-Coloring problem, we have 3 colors to assign to 50 vertices so that no adjacent node has same color. There might be more than one optimal solution for the problem. Therefore, RHC module performed better than in Four Peaks problem. More global optima means more chance that RHC restart landed in the global optima domain and found it. As long as RHC is restarted until optimal domain is found, RHC module worked. However, it is counter-productive because we cannot evaluate all possible states and do not want to waste resources on evaluate all possible solutions.

In this problem, we chose the number of vertices large enough to contrast the SA and GA performance. Overall, both modules produced excellent results. Although as the problem get more complicated and higher dimensions are introduced (more colors to choose) GA are more consistent than SA to find the optimal solution. In case of higher dimension and more than one possible global optima arise, RHC also produced good result due to probability of it landed on a global optima increase.

In figure 4, we ignore RHC fitness curve for now as its curve only showed performance of the best restart, however, RHC showed very promising result for K coloring problem due to higher chance of finding optimal solution. SA module can show good result but required more iteration, particularly due to our chosen exploration strategy. The more complicated the problem with more local optima, the more exploration is required for SA module. Regardless of how many iteration required until converged, SA module is still very efficient with computational cost, which is showed in table 2. SA required more hyperparameters tuning and as the problem become more complicated, we have to increase our initial temperature to allow the module to explore more.

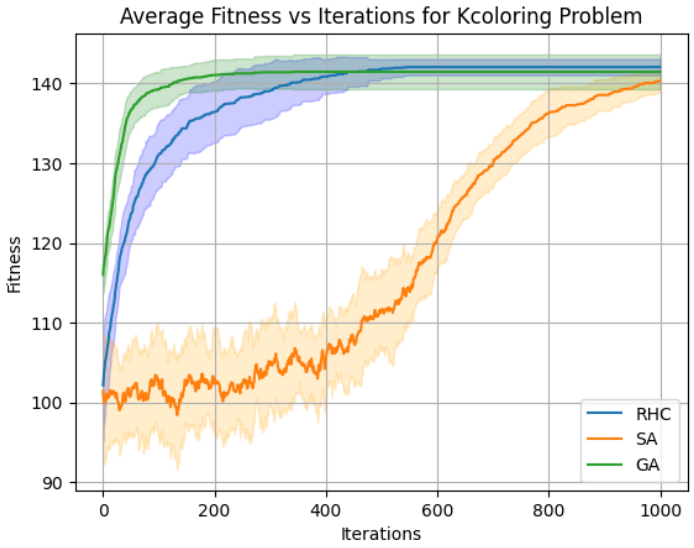


Fig. 4: Average best fitness for K Coloring problem

In term of performance on iterations, GA module is superior than the other two. It can find optimal solution for K-coloring problem with least iteration and evaluation. The fitness variance is also low, indicating that it can find optimal or close to optimal solution most of the time among our 20 repeated procedures. GA module is truly a robust solution for K-coloring.

| | Avg Best Fitness | Avg FEval | Training time |
|---|---|---|---|
| RHC | 142.0 | 21,881 | 52.3s |
| SA | 140.3 | 1,607 | 1.5s |
| GA | 141.5 | 13,100 | 4.5s |

TABLE II: Summary of performance metrics in Four Peaks

Similar to Four Peaks, we tune the similar hyperparameters for each optimization algorithm. Our final modules to solve K-Coloring are RHC with restart of 50, SA with initial temperature of 400, and GA with population size of 50 and mutation rate of 0.2. All three models achieve similarly good average fitness score. In term of FEvals, SA module evaluated least number of functions, followed by GA then RHC. Although GA module evaluated 13,100 functions (table 2), the time it took to train is only 4.5 seconds, compared to SA module with 1,607 functions but took 1.5 seconds to train. That indicated SA functions required more effort than GA functions. GA functions required less computational cost because it found convergence very quickly, which is showed on the green curved in figure 4.

RHC required most number of functions to be evaluated and took substantially longer time to train 52 seconds. This indicates that RHC can be used to solve K-coloring problem, given enough restart times and computational power, highlighting its inefficiency.

Because the fitness function does not specify the color, any color can be swapped with another color entirely, like a label swap without affecting the fitness, as long as no adjacent node carried same color. K-coloring problem can have many optimal solution. The is particularly helpful for RHC which found the optimal solution by chance.

In figure 6, when we increase the number of nodes as a way to increase the problem size. All three optimization modules are on par with each other with SA having a slight

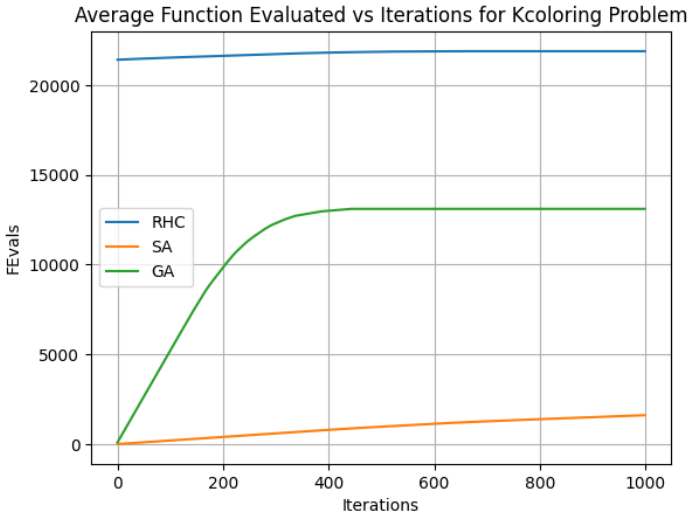


Fig. 5: Average Functions Evaluated for K Coloring problem

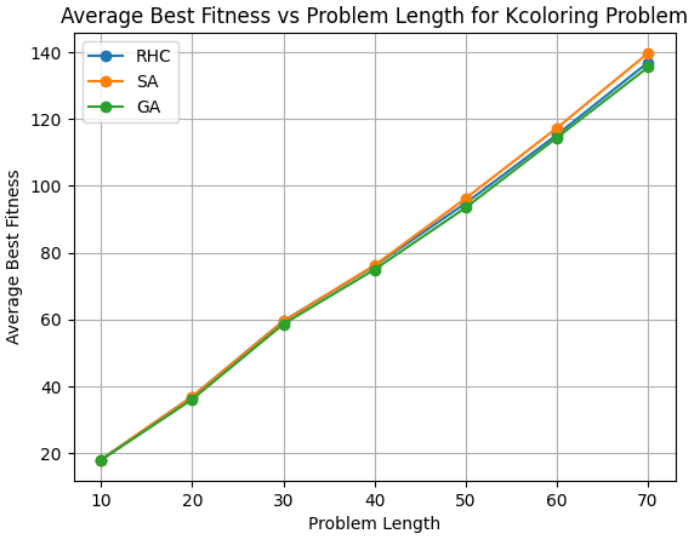


Fig. 6: Average fitness score for different problem size

edge. This indicated that all three algorithms can be used to solve K-coloring problem. However, their computational cost are vastly different. RHC required huge amount of computational power compared to the other two, as the problem size increased, RHC should not be considered.

SA is the most efficient module and required least computational cost. However due to its exploration characteristics and the existing of multiple global solution, SA sometime bouncing around even though it might already find the global optima. It is inefficient in that way but still the most efficient among the three algorithms. In addition, SA required hyperparameter tuning for every change in the problem size.

GA module is consistently good in solving K-coloring problem. It can achieve low bias and variance. It is not as efficient as SA module. However due to SA module is less efficient in K coloring. The use of GA module to solve K-coloring problem is superior.

## V. Neural Network Experiment

In the second part of experiments, we will re-visit an experiment from assignment 1. In this experiment, we will

use the traditional backpropagation process to update the weights of a neural network. Alternatively, we will update the weights of the neural network using three randomized optimization algorithms of interest and compared their results and performance.

The dataset using in this experiment is the white quality dataset. Our target variable is the wine quality score which is categorized into 3 classes: low, medium and high quality. The data is imbalanced with 33% in low quality, 45% in medium quality and 22% in high quality. In previous assignment, a neural network was able to handle the imbalanced data with accuracy score of 62%.

The dataset have 11 features with total of 4,898 records. All have been processed from previous assignment, there is no multicollinearity. The data is split into 80% used for training data and 20% used for testing. Five fold cross validation was used to assess the performance of model (through loss curve and learning curve).

Besides backpropagation, we also used three optimization algorithm to find optimal weights for the neural network. Each algorithm is run in 3 fold cross validation with different epoch number due to limited computational power. The same dataset is used to train each model.

## VI. Analysis of Neural Network optimization

In this assignment, we re-train the neural network using skorch library package with some minor changes. We increase our maximum epochs (numbers of iteration through the dataset) to 100 from 50, change the optimizer from adam to SGD. From previous assignment, we knew this should not make any significant improvement. In fact, our neural network trained with backpropagation resulted in 62% accuracy same as previous assignment. The current model handled imbalanced dataset well with not much different in f1 score across all classes(Table3). As number

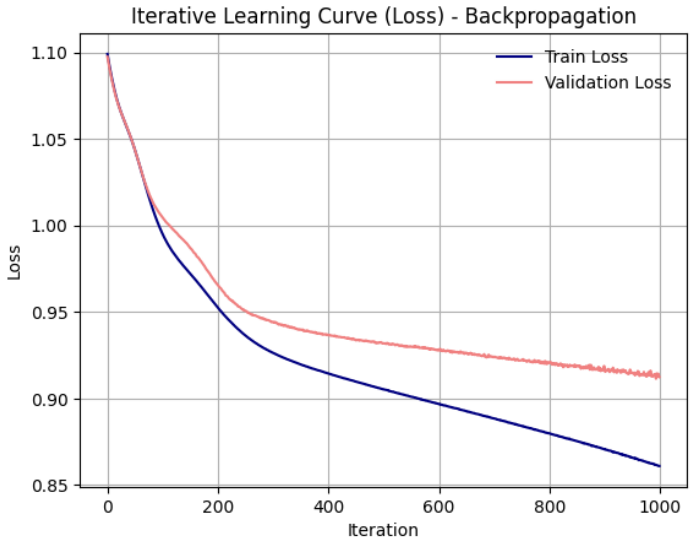


Fig. 7: Loss curve of backpropagation NN under different epochs

of iteration increase, loss decrease which is good. However, training loss and validation loss curve diverged indicating that it is overfitting, only training loss get better with more iteration (Figure 7). This is confirmed again in figure 8,

| | Recall | F1 |
|---|---|---|
| low | 62% | 65% |
| medium | 68% | 63% |
| high | 49% | 55% |
| accuracy | | 62% |

TABLE III: Neural Network with backpropagation model in wine quality dataset

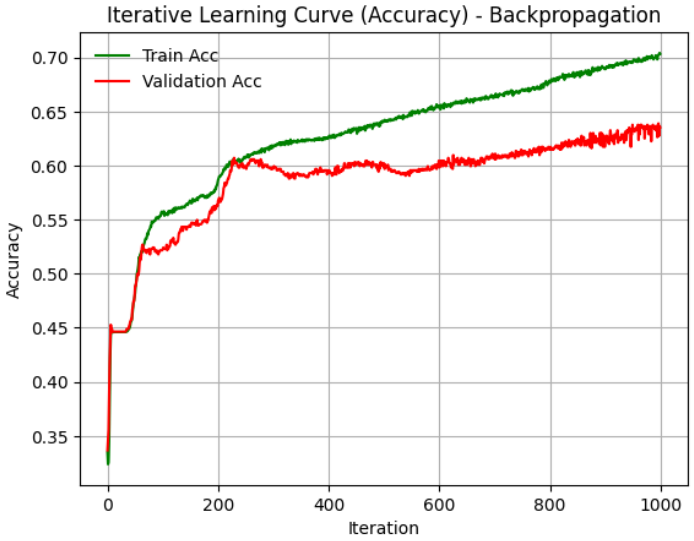


Fig. 8: Learning curve of backpropagation NN in different iteration

when both curve went sideway after 200 iterations. As the distance between training and validation curve increase, this is indication of overfitting, so our choosen number of iteration should not be more than 200. Using gridsearch to

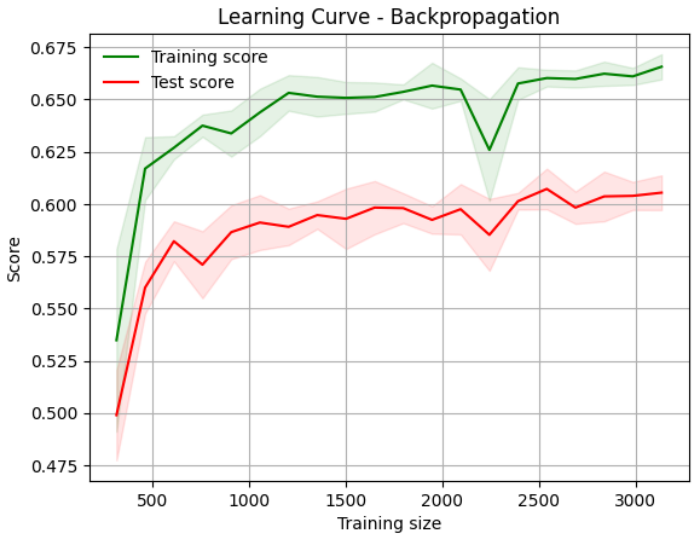


Fig. 9: Learning curve of backpropagation NN under different data size

fine tune the hyperparatermets, we decide to use 2 layers of 200 nodes for the neural network, the same numbers as previous assignment to make direct comparison. In fact, we have similar result, indicating that both model performed well. The learning curve under different size have gone side-sway after training through 1,000 training records and show no further improvement, as well as the distance

between tranining and validation score show no significant train indicating the model is not overfit.

### A. RHC

In a similar manner, we used the RHC module to update the weights of a neural network. The training time of the neural network with RHC is significantly higher than back-propagation and SA module, but faster than GA module. Based on the loss curve, the curve start diverging after 800 iterations, indicating where the model start overfiting. The performance improvement after 800 epoch decelerated, so we used epochs of 800 to train RHC module to update the neural network. It took significantly longer than training backpropagation with 200 iterations. Figure 12 showed

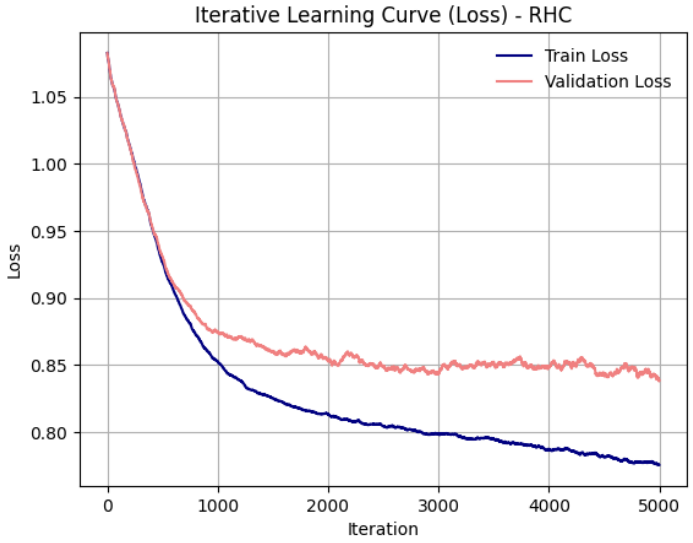


Fig. 10: Loss curve of RHC under different number of iteration

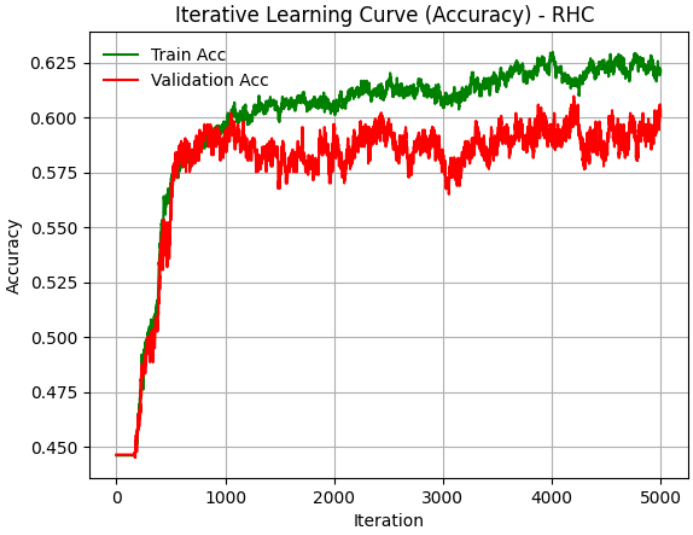


Fig. 11: Learning curve of RHC under different number of iteration

stable distance between 2 curves, indicating model is not overfit. Interestingly, updating neural network using RHC module does not require significant data. The number of data required to train stable RHC module is over 1,000,

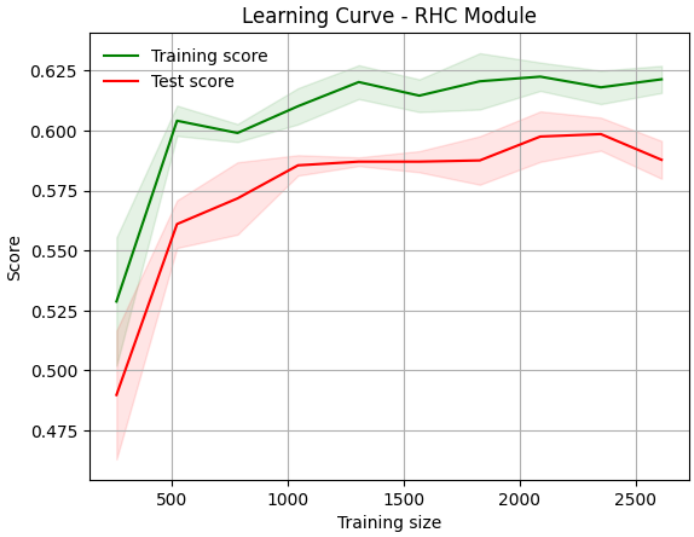


Fig. 12: Learning curve of RHC under different data size

which is somewhat similar to backpropagation model. RHC module have accuracy score of 57%. It however fall short of prediction for high quality wine, achieved only 0.25 f1 scored, compared to 0.6 and 0.62 for low and medium class respectively. Overall, RHC can be used to updated weights for neural network at a trade off of significant computational cost. Technically, RHC can find optimal weights given enough restart or iteration in this case.

### B. Simulated Annealing

Simulated Annealing is better at escaping local optima compared to RHC, and can explore a larger search space. However, SA required careful hyperparamters tuning. In figure 13, SA loss validation loss curve actually become worse after a number of iteration. Unlike RHC and back-propagation, their validation curve did not get worse. So we have to precisely pick the epoch number between 800 and 1200. The probability of accepting worse solution is controlled by temperature parameter, which is gradually adjusted after each iteration. Therefore, it is required hyperparameter tuning as well. In figure 14, the range that achieved high accuracy and low distance between training and validation curve is around 800 to 1000 iteration. Due to high computational cost, we decided to train with epoch of 200. The learning module showed two curve almost similar, suggesting good generalization. It is however does not achieve really high accuracy, particularly because of our limited iteration due to computational cost. SA module achieve much better result after 3000 training data size, which made sense because our SA module prioritize exploration at the beginning before exploitation. We can decrease the temperature faster to shift to exploitation priority if our data size is small. We kept the current temperature geometric decay schedule because we have sufficient data to do so.

Even with low number of iteration, SA module achieve good accuracy score. It however performed bad with prediction of high quality wine, indicating that SA module is bad with imbalanced dataset, similar to RHC and GA. The low accuracy score is probably due to low iteration number. Something we could have done better. Overall, SA module

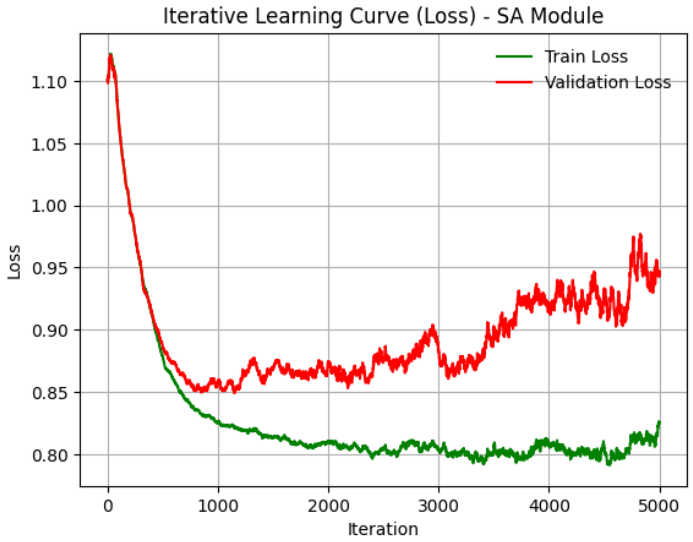


Fig. 13: Loss curve of SA under different number of iteration

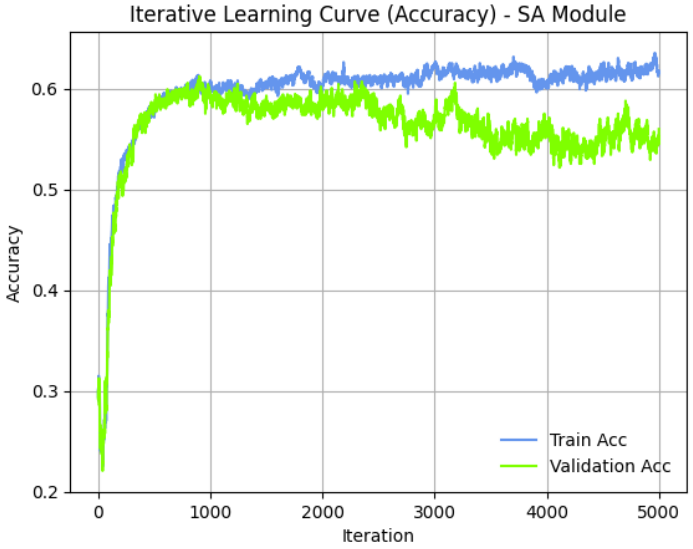


Fig. 14: Learning curve of SA under different number of iteration

is cable of finding optimal neural network weights with more training and our current model is underfit.

| | Class | F1 | Accuracy |
|---|---|---|---|
| RHC | low | 60% | 57% |
| | medium | 62% | 57% |
| | high | 25% | 57% |
| SA | low | 58% | 52% |
| | medium | 59% | 52% |
| | high | 0% | 52% |
| GA | low | 61% | 56% |
| | medium | 61% | 56% |
| | high | 26% | 56% |

TABLE IV: Neural Network with Randomized Optimization propagation in wine quality dataset

### C. Genetic Algorithm

GA module took the most computational power to train in our experiment. At iteration of 100, it already took us an hour to process. GA module is expected to be the most effective among the three modules for optimizing neural network weights due to its robust search capabilities and

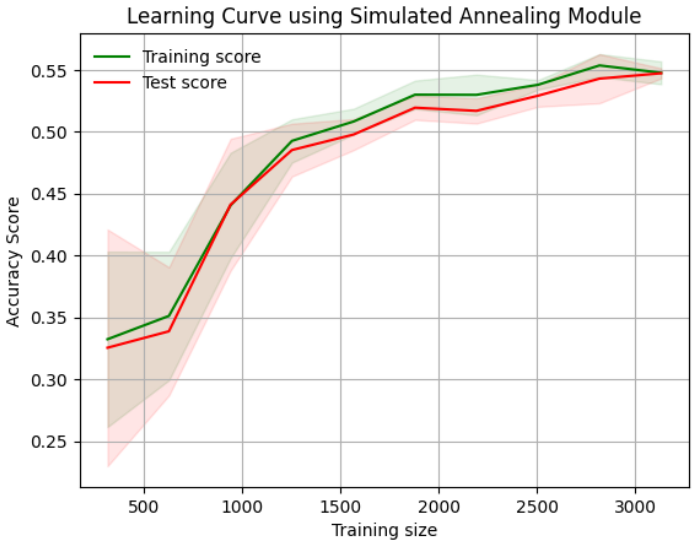


Fig. 15: Learning curve of SA under different data size

ability to explore diverse space. GA module however took so long to train and required significantly more memory. In figure 16 and 17, we saw that the distance between training and validation curve start widening, indicating 100 iteration might be the good enough and the model converged. However, we could not confirm further.

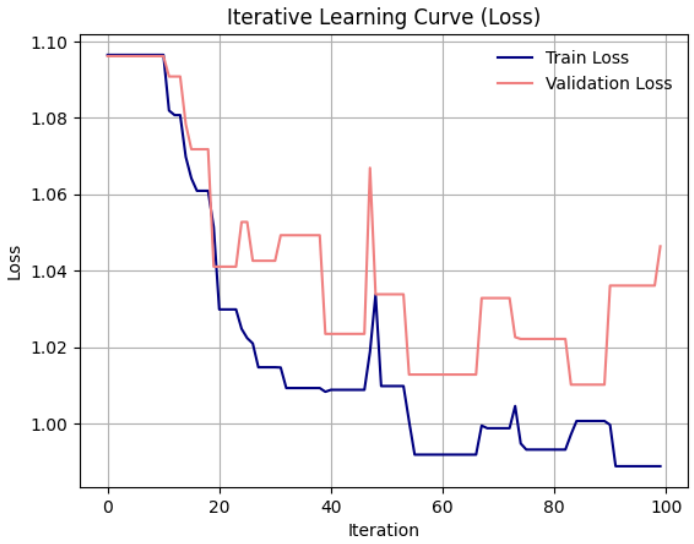


Fig. 16: Loss curve of GA under different iteration

After hyperparameter tuning, we decided to use GA module with 200 epochs, population size of 100, mating population of 50 and mutation size of 20. In figure 18, our learning curve converged around 3,000 training size, indicating that the module required significant data. It could also because of our decision on population size and mutation rate.

In figure 18, both training and validation learning curve are similar indicating that GA module generalized well, which is expected. In table 4, GA module achieved very good accuracy and f1 score. Although like RHC and SA, it does not handle minority class prediction well, so not good for imbalance dataset.

All 3 modules' accuracy results are lower than the back-

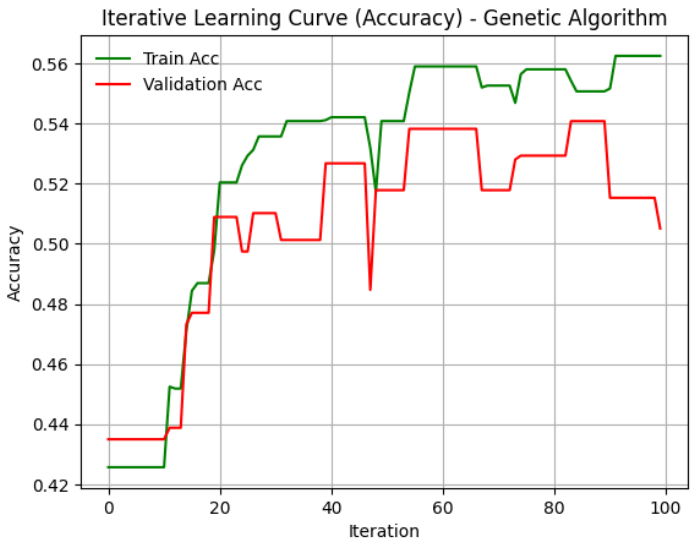


Fig. 17: Learning curve of GA under different iteration

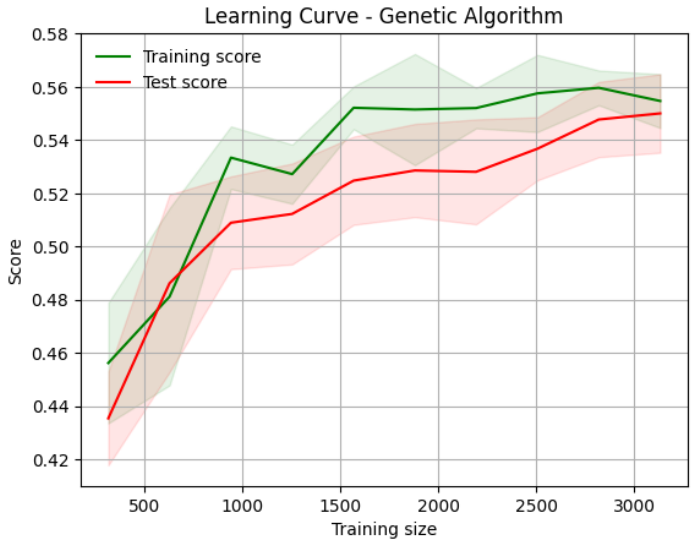


Fig. 18: Loss curve of GA under different training size

propagation module's, but not too far off, indicating that they can be used. It is however takes significant amount of time and resource to train which might not be possible in some cases. Our training of GA module with 200 iteration took nearly 3 hours, while backpropagation only took 15 minutes.

Overall, randomized optimization algorithm can be used as an alternative way to optimizing weights of neural networks at significant cost. GA module is the most extensive for this purpose but required most computational resources. SA module can also be effective but require extensive tuning despite that our result is underfiting. RHC module surprisingly perform well for our particular dataset, however, given a dataset that have more features and dimensions, we expect RHC module struggle.

## VII. Limitation

All our modules did not perform well with imbalance dataset. It is suggested that we applied SMOTE or other over sampling technique to transform our dataset to be more balanced. Our SA module is underfit and could have been better with more extensive hyperparameter tuning and more iteration training. Given more computational power, we can also train GA module under more iterations. In fact, GA module can also be trained in parallel to increase the speed.

## VIII. Conclusion

In conclusion, the experimental analysis of Four Peak and K-Coloring problem demonstrates that Genetic Algorithm (GA) are a robust optimization solution due to their ability to effectively explore the search space and avoid local optima. However, Simulated Annealing (SA) has shown to be a more efficient approach in Four Peak problem. It required fewer function evaluations and less training time, although, it demanded extensive hyperparameter tuning. In the case of K-Coloring, GA consistently outperform other methods, exhibiting low bias and variance, whereas SA is less efficient due to its tendency to bounce away from multiple global optima. Random Hill Climbing (RHC) as anticipated is the least effective due to its propensity to get stuck in local optima, although in simpler version of Four Peak and K-Coloring, RHC offer an alternative choice.

For optimizing neural network weights, while RHC, SA and GA present viable alternatices to traditional back-propagation, the latter remains preferred method due to its efficient gradient computation and faster convergence. Although, GA showed promising result in term of accuracy, it required significantly more training time and computational resources.

It is important to acknowledge the limitations in computational power during these experiments, which lead to underfitting in case of SA. Both SA and GA could benefit from increased iterations during training. In addition, GA might be trained faster if parallel training is implemented.

Finally, while alternative optimization algorithm such as SA and GA offer valuable insights and potential for specific application, backpropagation maintains its superior for neural network training due to its efficiency and performance. There has been more study and experiment to increase its efficiency as well. The findings highlight the importance of computational resources and promising alternative weight-optimization method that is genetic algorithms.

## IX. Resources

[1] *API Reference.* Scikit-learn. https://scikit-learn.org.
[2] UCI. Wine quality dataset https://archive.ics.uci.edu/dataset/186/wine+quality
[3] *API Reference.* pyperch https://github.com/jlm429/pyperch/.
[4] *API Reference.* mlrose-hiive https://github.com/hiive/mlrose.
[5] *Marian Tietz and Thomas J. Fan and Daniel Nouri and Benjamin Bossan and skorch Developers.* Skorch: A scikit-learn compatible neural network library that wraps PyTorch https://skorch.readthedocs.io/en/stable/.