CS7641 A2: Randomized Optimization

Tien Hoang
thoang30@gatech.edu

I. Introduction

In this assignment I explore four local randomized search algorithms, Randomized Hill Climbing (RHC), Simulated Annealing (SA), Genetic Algorithm (GA), and Mutual Information Maximizing Input Clustering (MIMIC). Those are employed to find optimized solutions in 4 problems: Neural networks (nn) for binary classification, continuous peaks, flip flop, and max K-colors.

In the neural network problem, the back-propagation is removed and replaced by one of the RHC, SA, and GA algorithm. The running time, training loss, and recall score are compared for each algorithm. In other three problems, the fitness score and running time are studied.

We take the advantage of the ”mlrose” library in python for our analyses and generated data: https://mlrose.readthedocs.io/en/stable/.

II. Neural networks

A. *Credit Card Fraud Detection Data*

Source: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

The dataset consists of credit card transactions made by European cardholders in September 2013. It covers a two-day period during which 492 fraudulent transactions were detected out of a total of 284,807 transactions. It’s important to note that the dataset is highly imbalanced, with the positive class (frauds) representing only 0.172% of all transactions.

Due to confidentiality concerns, the information about the original features or additional background data can not provided. The dataset exclusively contains numerical input variables, which result from a PCA transformation. Principal components V1 through V28 are the outcomes of PCA, with ’Time’ and ’Amount’ being the only features that were not subjected to PCA. ’Time’ represents the time elapsed in seconds between each transaction and the first one in the dataset, while ’Amount’ corresponds to the transaction amount. The ’Amount’ feature could be useful, for instance, in the context of cost-sensitive learning that depends on transaction amounts. All of the features are numerical type. The ’Class’ feature serves as the response variable and takes a value of 1 for fraud cases and 0 for non-fraudulent transactions.”

Why did we select this dataset? We chose it because it aligns with our research objectives, necessitating binary classification. Its inherent class imbalance provides a valuable opportunity for data preprocessing learning. Moreover, its practical relevance makes it well-suited for addressing real-world challenges.

B. *Handling imbalance data*

Sampling techniques are the most popular strategies to deal with imbalance data in the pre-processing stage. We can use ”under-sampling”, ”over-sampling” or combine two of them. In this assignment, I use under-sampling technique for the training data while keeping testing data untouched. A new sub-sample with balance number of fraud and non-fraud are created by randomly chosen the subset of fraud transactions. Particularly, we choose $test\_size = 0.2$, the training size is $(788, 30)$.

C. *Results in comparison*

Thanks to the NeuralNetwork function in the ”mlrose” library, we can setup the problem with different search algorithms: Randomized Hill Climbing (RHC), Simulated Annealing (SA), Genetic Algorithm (GA) (MIMIC is not available for selection). In the setup, we mostly the default parameters. The hidden nodes is chosen $[2, 2]$, because increasing the hidden nodes doesn’t provide any better results. The activation is ’tanh’ which performs the best (GA does not seem to work on other activation functions such as ’identity’, ’relu’, or ’sigmoid’).

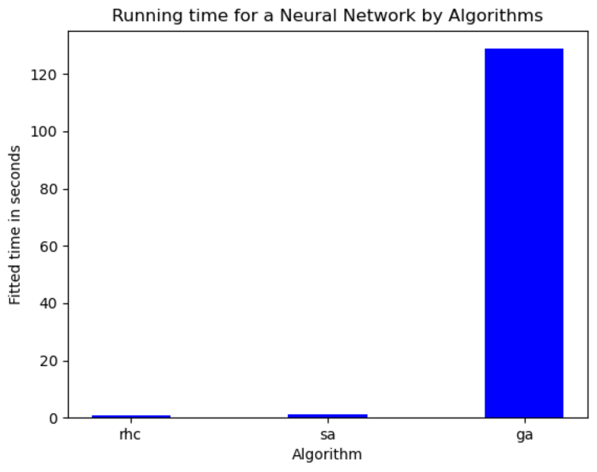

Fig. 1: Neural network running time for each algorithm

As shown in the figure 1, the SHC and SA algorithms have running time pretty quick (about a second) while the GA running time took lots longer. The results are agreeable with the fitness curves in figure 2: SHC and

SA have smaller started fitness and convergent quicker while GA takes many more iterations to converge.

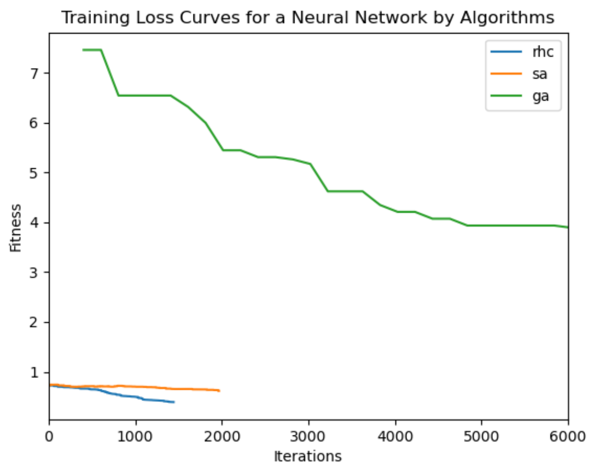

Fig. 2: Neural network fitness curve for each algorithm

Next, we want to compare how the neural network performs on each algorithm. It is kind of tricky to evaluate how well the model predictions on the very imbalance data. For example in out problem, the number of fraud is only 0.172%, then if the model predicts all negative which already have the correction of more than 99%.

Actual/Prediction	Positive Predictio	Negative Prediction
Positive Class	True Positive (TP)	False Negative (FN)
Negative Class	False Positive (FP)	True Negative (TN)

TABLE I: Confusion matrix

In our problem, we use the recall score for appropriate measure of the imbalance data predictions. The formula to calculate the recall score as following (from confusion matrix in Table I):

$$Recall = TruePositive/(TruePositive + FalseNegative)$$

Figure 3 shows the recall score for each algorithm with both training and testing data. It is worth to mention again that the testing data is original imbalance data. The scores are comparable for all three algorithms. The testing scores are above 90% for all of them.

III. Optimization Problems

We apply the 4 algorithms, RHC, SA, GA, and MIMIC, to 3 optimization problems: , continuous peaks (cp), flip flop (ff), and max K-colors (mkc). The data are generated from the 'mlrose' libarary with the functions: 'ContinuousPeaksGenerator', 'FlipFlopGenerator', and 'MaxKColorGenerator' respectively. Unless otherwise specified we run each algorithm ('RHCRunner', 'SARunner', 'GARunner', 'MIMICRunner') with 1,000 iterations.

Those problems below are inspired from the 'mlrose' library: https://mlrose.readthedocs.io/en/ stable/source/fitness.html

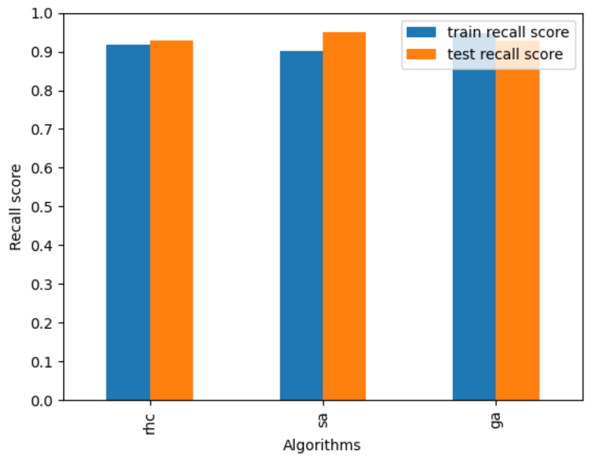

Fig. 3: Neural network recall score for each algorithm

A. *Continuous peaks (cp)*

The Continuous peaks fitness function is best suitable for use in bit-string (discrete-state with $max\_val = 2$) optimization problems. In the Continuous Peak optimization problem, the goal is to find the peak or global maximum of a continuous fitness landscape. Evaluates the fitness of an n-dimensional state vector x, given parameter T, as:

$$Fitness(x, T) = max(max\_run(0, x), max\_run(1, x)) + R(x, T)$$

where:

1) $max\_run(b, x)$ is the length of the maximum run of b's in x
2) $R(x, T) = n$, if $(max\_run(0, x) > T$ and $max\_run(1, x) > T)$
3) $R(x, T) = 0$, otherwise.

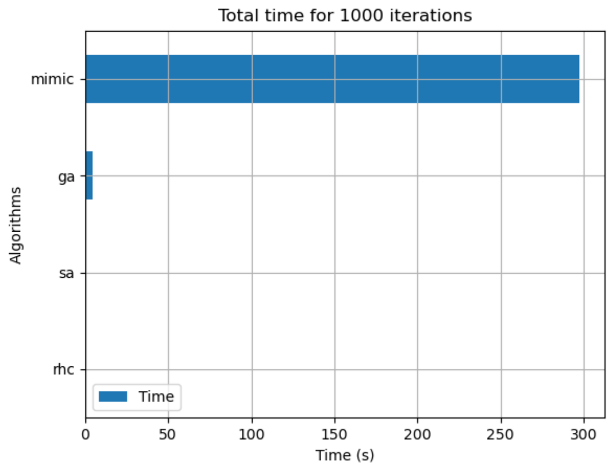

Fig. 4: Continuous peak running time for each algorithm

In the figure 4, the time to run 1000 iterations for RHC and SA are pretty quick, for GA is a longer, but for MIMIC is much longer.

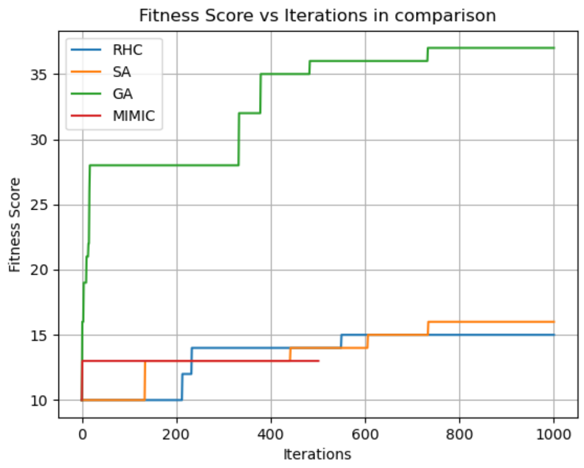

Fig. 5: Continuous peak fitness for each algorithm

Figure 5 shows the fitness of each algorithm as a function of iterations. The MIMIC get the the plateau quickly after one iteration. The fitness of RHC and SA is similar to each other and increases slowly with growing numbers of iterations. In the meantime, the fitness of GA is about three time bigger and increasing as well.

B. *Flip flop (ff)*

The Flip Flop fitness function is best suitable for use in discrete-state optimization problems. In the Flip Flop optimization problem, the goal is to find the optimal ordering of a sequence of binary bits (0s and 1s) in a way that minimizes the number of adjacent bits that are the same. Evaluates the fitness of a state vector x as the total number of pairs of consecutive elements of x, (x_i and x_{i+1}) where $x_i \neq x_{i+1}$.

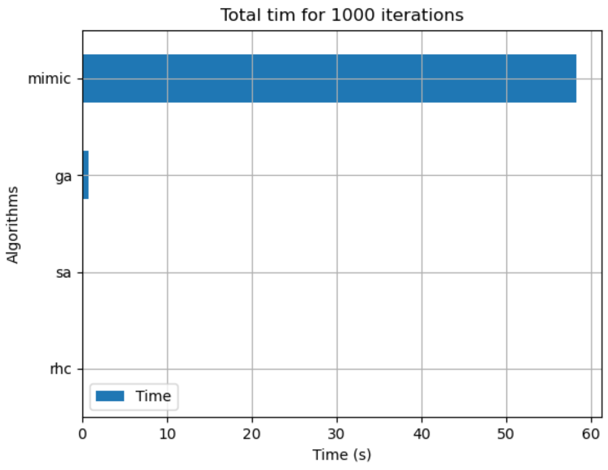

Fig. 6: Flip flop running time for each algorithm

The running time for the Flip flop problem is in the similar manner with the Continuous peaks (Figure 6. MIMIC still takes the longest running time, then GA and SA/RHC. However, the running time for MIMIC is reduced 5 time compared with the Continuous peak problem.

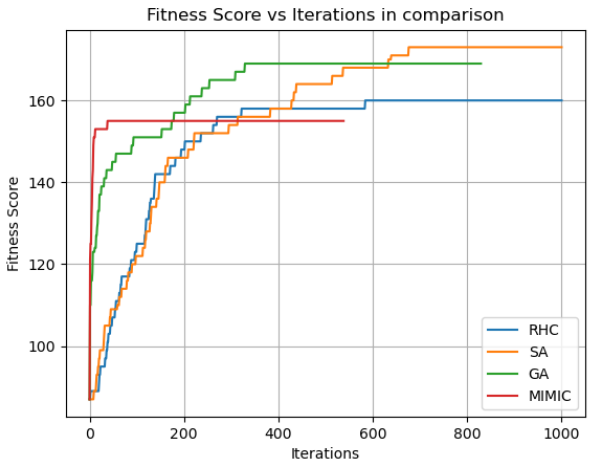

Fig. 7: Flip flop fitness for each algorithm

Figure 7 shows the fitness for 4 algorithms together. The maximums of the fitness are comparable while MIMIC obtains it quickest.

C. *Max K-colors (mkc)*

The Max k-color problem is an optimization problem that involves coloring the vertices of a graph with a maximum of k colors in a way that no two adjacent vertices share the same color. Evaluates the fitness of an n-dimensional state vector $x = [x_0, x_1, \ldots, x_{n-1}]$, where x_i represents the color of node i, as the number of pairs of adjacent nodes of the same color.

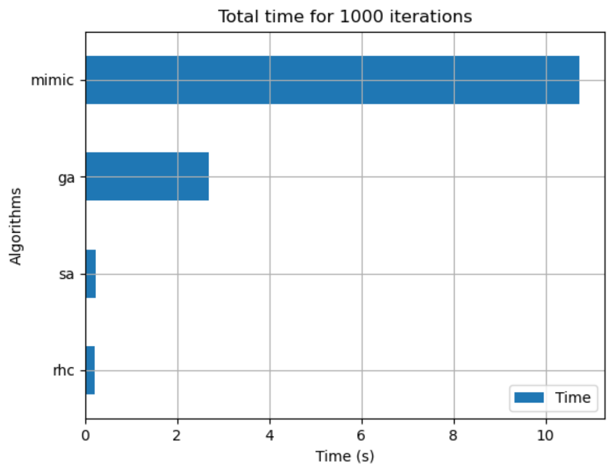

Fig. 8: Max K-color running time for each algorithm

As above, figure 8 shows the running time for 1000 iterations of 4 algorithms. In this graph, it clearly shows the ratios of those magnitudes as MIMIC still taking longest time, then GA, then SA and RHC. The MIMIC running time also 6 times smaller than the Flip flop problems.

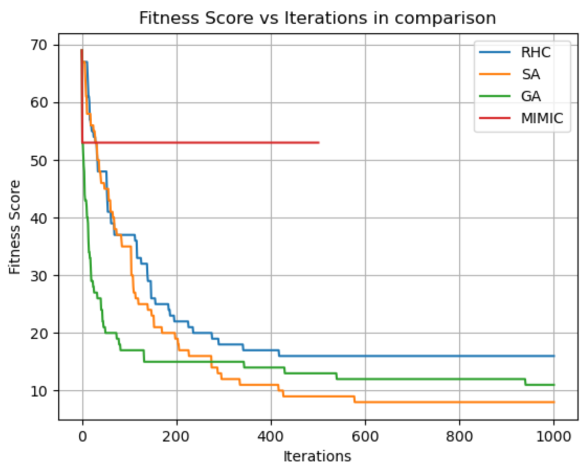

Fig. 9: Max K-color fitness for each algorithm

Figure 9 shows the fitness of these algorithms. It is noticeable that the fitness (for the Max k-color problem is going down as the number of iterations increase (find minimum instead of maximum). The fitness (in this case should be referred as "loss function") of the GA, SA, and RHC are naturally reduced to the plateau (after around 500 iterations). The fitness of the MIMIC does not changes after the first iteration.

IV. Conclusion

In this section, we discuss and summarize 4 local randomized optimizers applying to our domain problems.

1) Randomized Hill Climbing (RHC) is a simple and intuitive optimization algorithm used to find the optimal solution to a problem. It is a local search algorithm that starts with an initial solution and greedily updates its weight vector in the direction of the steepest hill. RHC appears to be a good choice for quick exploration of a solution space with a unit space complexity O(1) and a linear time complexity O(n) with respect to the number of its neighboring states. However, it is not guaranteed to find the global optimum, as it may get stuck in a local optimum.
2) Simulated Annealing (SA) is a gradient-based technique combining both hill climbing and random walk that yield both efficiency and completeness. It is highly effective in finding global optima because it can escape local optima by occasionally accepting worse solutions early in the search. In our study, the SA algorithm also has time complexity O(n) (n is the problem size).
3) Genetic Algorithm (GA) works by evolving a population of candidate solutions over multiple generations to reach an optimal or near-optimal solution. As our studies show, GA will take considerable longer running time for all of our problems. The time complexity of a Genetic Algorithm (GA) depends on the size of the population (N), the length of the genetic representation (L), the complexity of the fitness function (F), and the number of generations (G). So that, the time complexity is O(N*L*F*G).
4) Mutual Information Maximizing Input Clustering (MIMIC) is a machine learning technique and optimization algorithm used for feature selection and data clustering. MIMIC is useful in a high-dimensional dataset with many variables. In our problems above, MIMIC is not very useful and usually takes longest time to run. The time complexity of MIMIC algorithm can vary significantly depending on factors: the size and complexity of the dataset, the choice of the parameters.