Assignment#2, CS7641

Georgia Institute of Technology, Fall 2020

Asif Rehan, arehan7@gatech.edu

Introduction:

There are two main parts in this assignment#2. First, four local random search algorithms Randomized Hill Climbing(RHC) , Simulated Annealing (SA), Genetic Algorithm (GA) and Mutual-Information-Maximizing Input Clustering (MIMIC) were used on three optimization problem domains, OneMax, FlipFlop and FourPeaks. In the second part, a Neural Network model is developed on the Pima Diabetes dataset. The Neural Network model weight optimization is done using RHC, SA, and GA. For this assignment, MLROSE_Hiive package was used throughout.

Randomized Optimization Algorithms

Randomized Hill Climbing(RHC)

It does Hill Climbing when a better fitness value is found until a set time expires. It works very well for a convex search hyperplane with little memory requirement but may take longer sometimes.

Simulated Annealing (SA)

This algorithm take idea from metal annealing and works well for unconstrained and bound-constrained optimization problems. It accepts a new fitness region based on the probability which relates to the decreasing temperature. This often accepts a new region to move on to based on this randomness which lets it get out of a local optimum trap.

Genetic Algorithm (GA)

This algorithm uses evolutionary biology concept and finds a better solution by mixing them in a generation and creates new offspring solutions which hopefully creates even a better one as better solutions like genes stays within and best solutions survive. It is a computationally more expensive for time and memory but works very well in practice for many randomized optimization situations.

Mutual-Information-Maximizing Input Clustering (MIMIC)

This algorithm creates a sample from a distribution from the parameter space and keeps the best ones above the defined threshold of fitness value and repeats the same process within this sample until the global optima is identified. It retains the internal structure of the function being optimized which is not present in SA, RHC, and GA.

Part 1: Optimizing Fitness Functions

1. FourPeaks

Four Peaks optimization problem [Hayes 2019] computes the fitness of a n-dimensional state vector x , given parameter T, as:

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

Four peaks problem has two local optima which has a wide basins of attraction. These local optima tries to trap SA and RHC. The two other global optima are more likely to be discovered by GA.

Choosing Best Parameter for Each Algorithms:

To be able to identify the proper parameter settings for the 4 optimization algorithms, a direct approach was taken instead of automated Grid Search to get more hands-on experience. For each algorithm, a range of parameter values were selected and then optimization is run. For each algorithm, fitness vs iteration curve and time vs iteration curves were plotted and manually inspected. For RHC, upto 10 restarts were allowed since it may get stuck at a local optima.

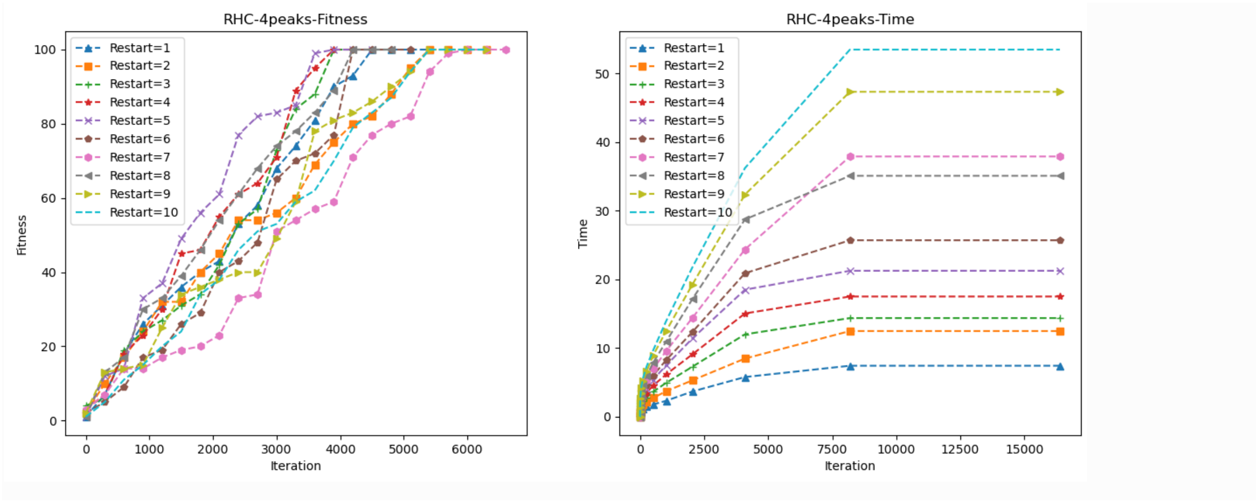


Figure 1: RHC Paramter Selection for 4 peaks problem

For RHC, restart=5 found the best fitness value the fastest among all param configurations. Also its runtime increased logarithmically with restarts number. This is intuitive that, more restart takes more time.

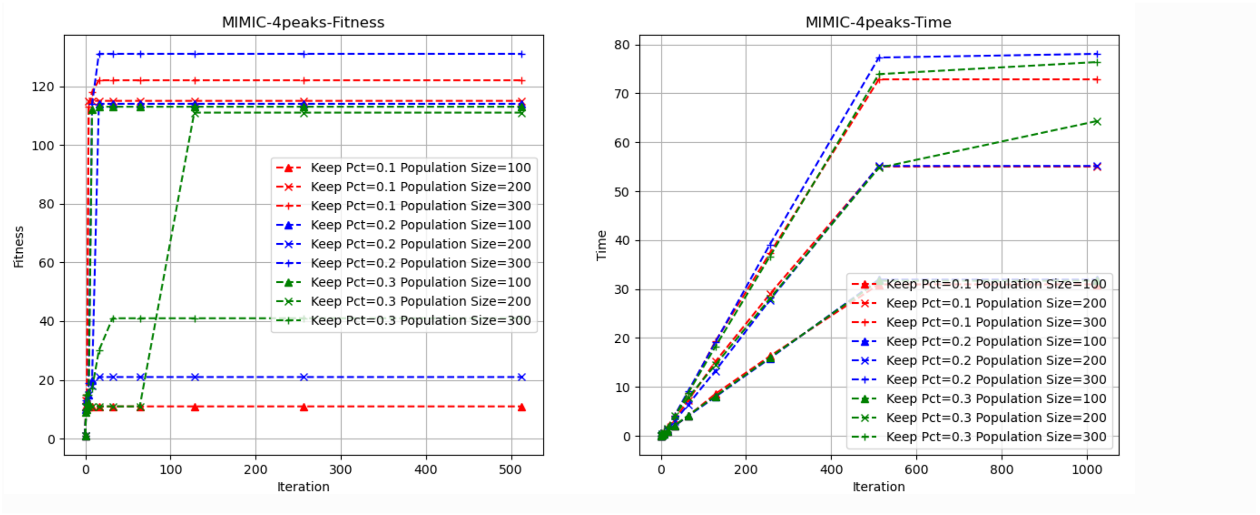


Figure 2: MIMIC Paramter Selection for 4 peaks problem

Similarly, for MIMIC, Population size=300 and keep_size=0.2 reached the best fitness score the first within just a few iterations. MIMIC runtime relationship with the parameter combinations are not clear from data and plot.

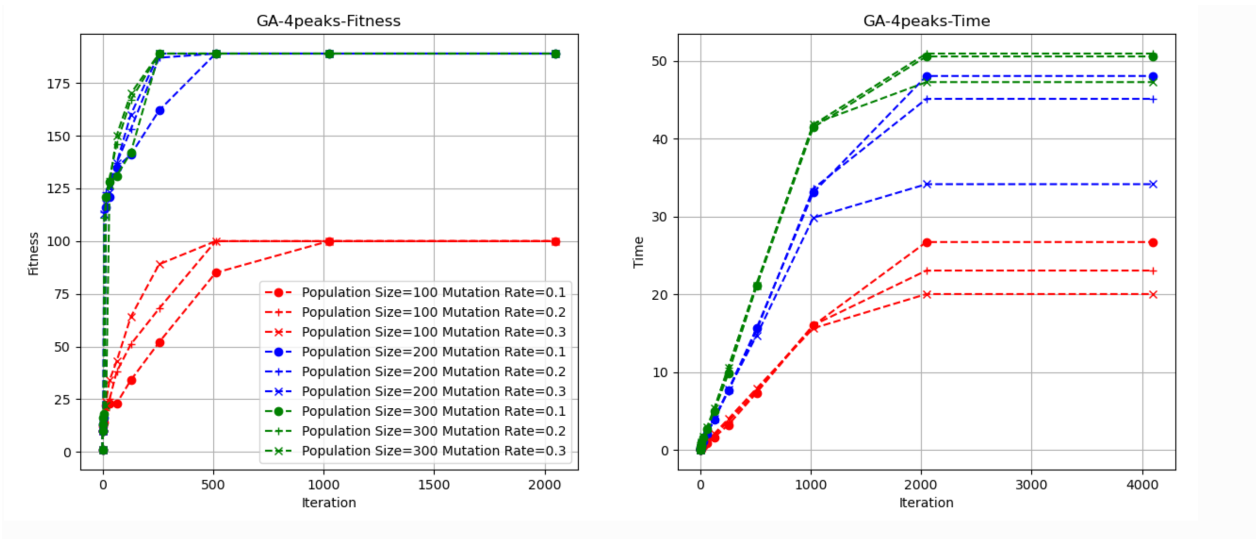


Figure 3: GA Paramter Selection for 4 peaks problem

For GA, mutation_rate=0.3 and population_size=300 reached the best fitness score the first among other competing parameter settings. From Time vs Iteration plot, runtime increased with higher population size and lower mutation rate.

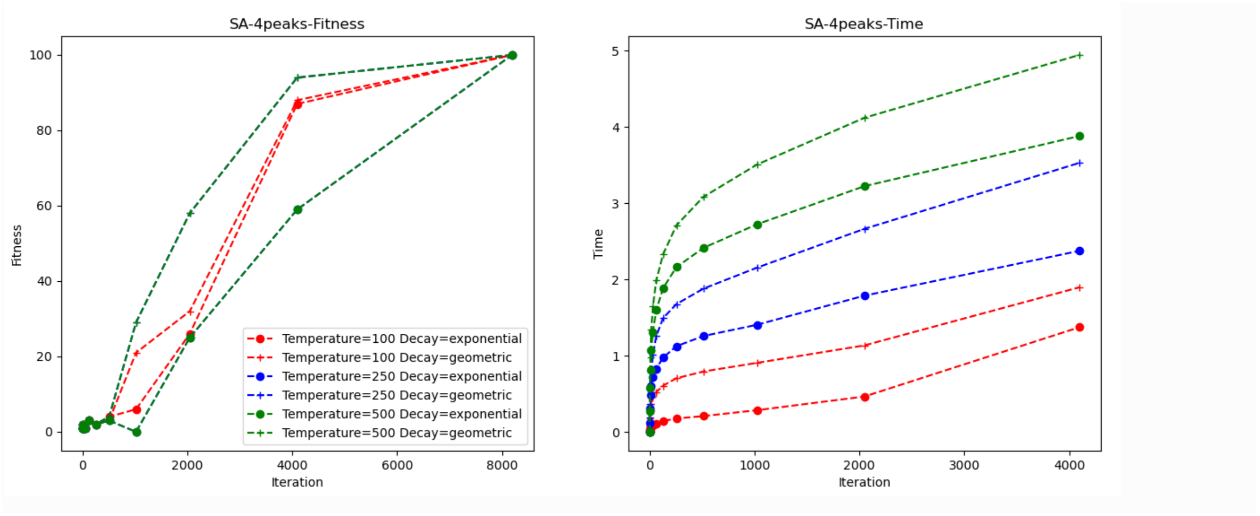


Figure 4: SA Paramter Selection for 4 peaks problem

For SA, intial Temperature=250 and Geometric Decay scheduler with default decay=0.99 and min_temp=0.001 reached the best fitness score fastest. Though apparently for a bug, the Temperature= 500 covered the one for 250. From Time vs Iteration plot, runtime increased with higher initial temperature and for geometric decay over exponential decay function.

Comparison of the Algorithms with their Best Parameters:

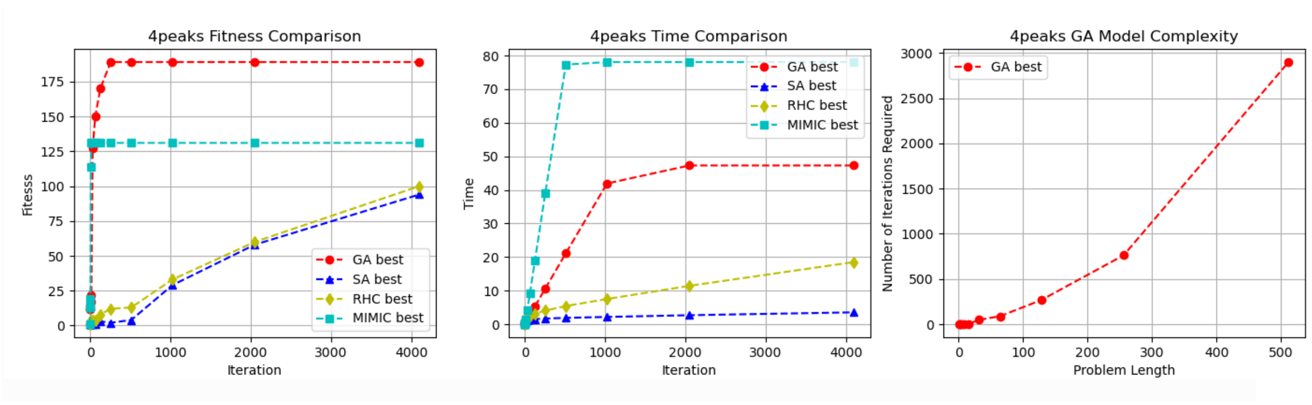


Figure 5: Algorithm Comparison for 4peaks problem

Finally, after the best models from RHC, GA, SA, and MIMIC are chosen, their fitness curves show, GA reached the highest fitness value over 175 but took less time than second best performer MIMIC. It is computationally expensive to compute iterations of MIMIC but it often takes significantly less iterations compared to other. The right chart shows, the number of iterations required for GA to reach the best fitness value for different 4peaks problem length in a geometric progression from 1, 2, 4, 8, 16, 32, 64,…, 512. In the chart, GA requires exponentially more iterations for increasing length of 4 peaks problem.

This problem shows the strength of GA. These local optima did trap SA and RHC. The other global optima were successfully discovered by GA.

2. FlipFlop

For a state vector x with n-dimensions, it computes the number of pairs of consecutive elements in x, (x_i and x_{i+1}) where $x_i \neq x_{i+1}$ [Hayes 2019].

Choosing Best Parameter for Each Algorithms:

For RHC, restart=7 found the best fitness value the fastest among all param configurations. Also its runtime increased logarithmically with iteration number. For GA, mutation_rate=0.1 and population_size=100 reached the best fitness score the first among other competing parameter settings. From Time vs Iteration plot, runtime increased with higher population size and lower mutation rate. For SA, initial Temperature=500 and Geometric Decay scheduler with default decay=0.99 and min_temp=0.001 reached the best fitness score fastest. For MIMIC, population_size=300 and keep_size=0.3 reached the best fitness score the first within just a few iterations.

Comparison of the Algorithms with their Best Parameters:

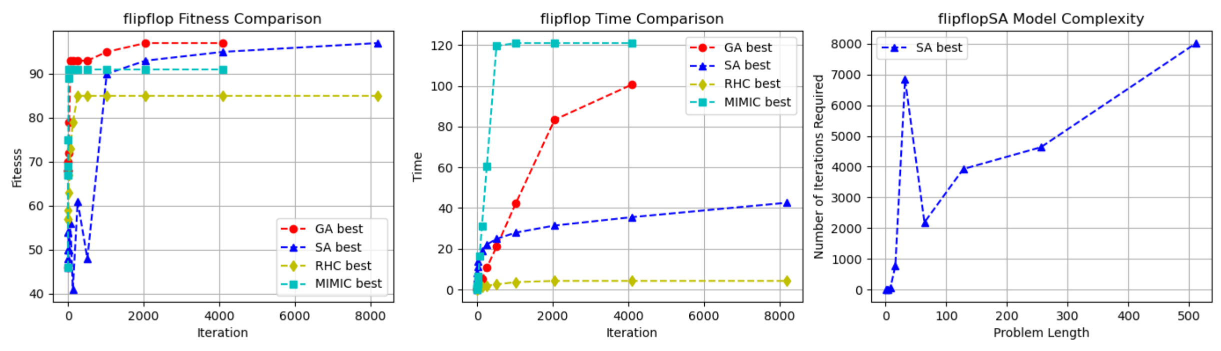


Figure 6: Algorithm Comparison for FlipFlop problem

Since flipflop is pretty much a straightforward fitness function, it takes less effort than FourPeaks to find the optimum for most algorithms. Though GA was the fastest but still SA actually performed better but took more iterations, in left above. But SA iterations still took much less time than GA since GA iterations are more computationally expensive. On the right above, over a range of problem length from 1,2,4,…512, SA took linearly more iterations to optimize based on the problem length. There was a little bump at problem length=32 which might be a bug.

For a simple problem like this, it is clear that, SA can outperform others here as the problem requires finding the optimal state vector when each of the values are alternating so the maximum fitness value is found. Compared to more complex algorithms like MIMIC and GA, SA successfully found the optima in shorter time though the number of iterations were much higher.

3. OneMax

One max problem [Hayes 2019] computes the fitness for a N-dimensional vector representing the state $x = [x_0, x_1, \ldots, x_{n-1}]$ as: $Fitness(x) = \sum_{i=0}^{n-1} x_i$. One max has a relatively simpler structure. So SA and RHC can find the optimum fitness value quickly. But in spite of that, MIMIC is expected to converge in the shortest iteration due to its ability to carry over the internal knowledge of the fitness function through iterations which is lacking in simpler random approaches like SA and RHC.

Choosing Best Parameter for Each Algorithms:

For RHC, restart=9 performed the best in iteration number and runtime among all param configurations. Also, its runtime increased logarithmically with iteration number. For GA, mutation_rate=0.3 and population_size=300 reached the best fitness score the first among other competing parameter settings. For SA, initial Temperature=100 and Geometric Decay scheduler with default decay=0.99 and min_temp=0.001 reached the best fitness score fastest. For MIMIC, population_size=300 and keep_size=0.2 was the winning parameter configuration.

Comparison of the Algorithms with their Best Parameters:

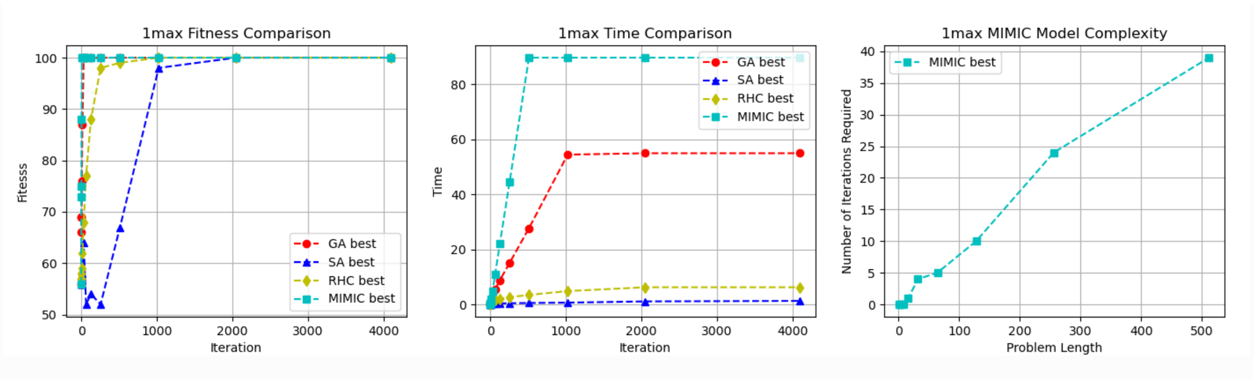


Figure 7: Algorithm Comparison for OneMax problem

In the chart above, OneMax problem shows that the simple problem allows all the four algorithms to find the optimum within 2000 iterations. But the biggest strength of MIMIC is found here that, MIMIC found the optimum in just a handful of iterations even though its iterations are more complex than RHC and SA. Still MIMIC finds the optima in way less iterations than them. Though RHC and SA took less time for their 4000 iterations to find the max, MIMIC took much longer, again showing each iteration of MIMIC is more complex as it samples from the search region based on a probability and keeps the top performing ones. It could be also argues that it shows the strength of RHC, taking around 800 iterations but nearly as little time as SA. But MIMIC's strengths and weaknesses are more interesting as the simplicity of the problem already explains why RHC was doing so well.

In the right chart above, the problem length was varied from 1,2,4,8,…,512 again. For OneMax problem, MIMIC's took linearly more iterations based on the length of the problem. This shows that by retaining the internal structure of the problem and reusing it in the iterations behind the scene, MIMIC does not require exponential more iterations as the problem size grows.

Part 2: Optimizing Neural Network Weights

Data

From assignment#1, Pima Indian Diabetes dataset was used [1]. The dataset comes with different input variable with medical condition indicators such as age, BMI, insulin etc and the Outcome column, which is 1 if the target had diabetes or 0 otherwise, making it a binary classification problem. There are 768 records with 8 variables and 1 label column. This data is a bit imbalanced, there are 500 false (~65%) and 268 (~35%) true diabetes indicators. Visually no outlier was detected, and all features were numerical, so no feature engineering was required. The scatterplot shows there are a few outliers handled by preprocess step with standardization.

Methodology

Benchmark: In Assignment#1, a Neural Network model was built. No feature engineering or outlier removal was required as the data is mostly clean, the dataset was split into a 70% training and 30% testing test. That model was built using Scikit-Learn's MLPClassifier which used L2 normalization parameter and Adam.

But since for this Assignment#2, Randomized Optimization algorithms in MLROSE Hiive did not offer as many parameters as Scikit-Learn did. So instead, the backpropagation-based Gradient Descent (GD) in MLROSE Hiive was used as the benchmark to compare against other Randomized Optimization algorithms RHC, SA, and GA. But much of Scikit-Learn library was still useful to set up the experiment.

Neural Network Model: The training dataset was preprocessed as standard normal distribution. A Neural Network classifier using MLROSE class with hidden layers of [12,6,4,2] nodes at each layer was defined. Each algorithm was allowed maximum 100 attempts to optimize before early stopping before reaching the max iteration number.
Parameter Search: Now to find the best parameters for each of the optimization algorithms, ScikiLearn's GridSearch was used. Each algorithm was evaluated usig F1 score which is recommended for the slightly imbalanced dataset being a balanced parameter of harmonic mean of Recall and Precision. After each of the algorithm's best parameters were found using the 5Fold Stratified Cross Validation, the Neural Network model is trained using these parameters and fitted with the whole training data to see the prediction quality.

Algorithm	Best Params
GD with backprop	Activation=tanh, learning_rate=0.05, max_iters=2000, clip_max=10000000
RHC	Activation=sigmoid, learning_rate=0.5, max_iters=3000, restarts=10, clip_max =10
SA	activation=tanh,learning_rate=0.5, max_iters=3000, schedule=Geomeric Decay init_temp=1, decay param=0.99, min_temp=0.001, clip_max=10.0
GA	activation=tanh,learning_rate=0.5,max_iters=3000,mutation_prob=0.3,pop_size=100,clip_max=10.0

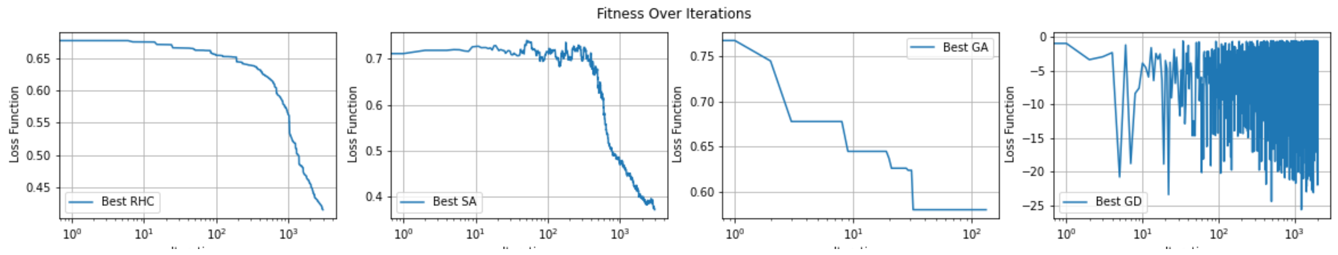
For benchmark GD with backprop, max 2000 iterations, Learning Rate=0.05, and a high value for a weight 10000000 (clip_max param) was allowed. For GD learning rate was small to let it make smaller steps size. Hyper parameter search showed, Tan Hyperbolic was the best activation function for GD with the best Training F1 score of 0.65 and accuracy of 0.74 over Rectified Linear Unit (ReLu) and sigmoid function.

For RHC, max 3000 iterations were allowed, Learning Rate=0.5, and clip_max=10 was used. Out of a range of 1-10 restart opportunities for RHC, it performed best with the most 10 opportunities to restart when stuck. The best activation turns out to be sigmoid over ReLu and TanH. Apparently more opportunities for RHC to restart the search at a new random search region could improve the performance. The tuned parameters had Training F1 score of 0.80 and accuracy of 0.74 over Rectified Linear Unit (ReLu) and sigmoid function.

For SA, TanH was the best param over other activation functions, with learning rate=0.5, and a Geometric Decay function with initiation temp of 1 and decay param=0.99 and min temp=0.001. The best accuracy after refitting to the entire train data was 0.86 and F1 score of 0.79.

For GA, population size=100 performed better than 500, and mutation probability of 0.3 was better than 0.5 and TanH was the best activation. This best GA parameter yieled accuracy of 0.71 and Training F1 score of 0.55.

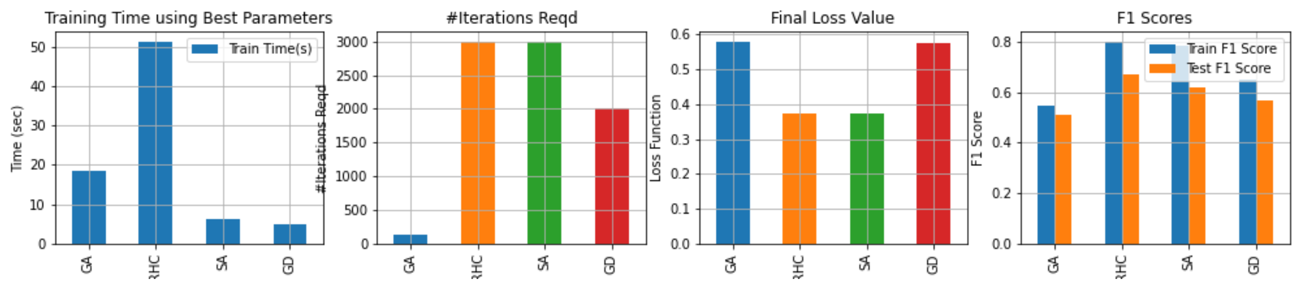
For each of RHC, SA, and GA the loss curve below show they all kept going down fast while GA keeps hitting a plateau and then it goes down in steps likely due to generations of evolutions sometimes take time to find best offspring solutions to evolve for better solution. Gradient Descent on the other hand, keeps a different loss function from the other three which is negative and oscillates as backprop-based gradient search tries to step forward and backward thinking it is a convex surface. This is one reason a differentiable activation function like tanh and sigmoid are more suitable for backprop.



The final performance is shown in the table below.

```
 Algorithm  Train Time(s)  #Iterations Reqd  Final Loss Value          Train F1 Score  Test F1 Score
        GA      18.366125               133          0.580466                0.546512       0.513158
       RHC      51.272870              3000          0.374693                0.801075       0.670659
        SA       6.294598              3000          0.372197                0.785311       0.618182
        GD       4.983852              2000          0.577321                0.647959       0.566265
```

Fitting the RHC took the most time (51sec) since it is a simpler approach and takes up all the allowable iterations (3000) just like SA. They both have simpler search strategy. GD was fastest to yield a quick answer in 5 seconds though. But again, with a different param setting GD could be better performing since it is highly susceptible to parameter selection just like to other ones. The final loss value for each of the algorithms, showed the final loss value of 0.37 for RHC and SA which also performed better than GA and GD in terms of Train and Test F1 scores.



Both RHC and SA did perform better in terms of F1 score, but they also overfit significantly. Interestingly, GA just got stuck at 133 iterations while all others ate up 3000 (SA, RHC) and 2000 (GD) iterations. Most likely, GA needed more attempts before the early termination of the process. This could easily show GA being one contending with RHC and SA in terms of F1 scores but at that rate GA would likely take much longer than RHC which took the longest to optimize the Neural Network weights.

There could be two types of training algorithms for the weights for a neuron. First is to minimize the error between predicted and actual. Here prediction is true if activation >= threshold. This type of perceptron-based learning works best for linearly separable data and guarantees finite iterations.

Second type is Gradient Descent algorithm which minimizes the error between the activation function value and the actual value. It has the similar nature as Least Square Regression. It is robust because it uses calculus. It is useful to differentiate based on activation function, because it has weights which makes it differentiable. On the other hand, perceptron-based error function is not differentiable with relation to weights. This is functions like sigmoid function which looks like a step function, continuously differentiable with a nice differential form performs better with Gradient Descent. In our Pima dataset, it is hard to claim that the weights are differentiable.

So, it is possible to find much better weights which performed better by using a Randomized Optimization algorithms instead of using a backpropagation-based Gradient Descent algorithm given proper parameter search region.

Conclusion

From part1 of analyzing the 4 Randomized Optimization algorithms, it was observed that GA and MIMIC iterations are more complex and time consuming. But unlike GA, which is more exploratory in nature, MIMIC uses the probability distribution of the more likely regions in the problem horizon which lets it reach the optimum in less iterations than others. GA takes exponentially more iterations with FourPeaks problem length. Simpler algorithm such as RHC and SA are good choice for problems whose fitness function is straightforward. SA and RHC may take more iterations for these problems but may still be faster than GA and MIMIC due to their each simple iterations.

From part2, it is possible to find better parameters for a problem like finding best weights for a neural network model using randomized optimization algorithms which does not require the weights to be differentiable. In a random dataset like the one used, it is unknown in advance if the activation function can be assumed differentiable. Randomized Optimization algorithms such as RHC, SA, and GA, do not have a limitation like the benchmark GD. So instead using Randomized Optimization is a powerful tool at hand. The results also showed the SA and RHC performed better than the backpropagation based GD.

Overall, finding when to stop iterating so the weights do not overfit is a real problem with randomized optimization. But even more importantly than that, determining the best parameters for all the four algorithms are non-trivial as they take significant time and domain knowledge to suggest what range of parameters may work best for a situation.

References:

1. Pima Indian Diabetes dataset: kaggle.com: https://www.kaggle.com/uciml/pima-indiansdiabetes-database?select=diabetes.csv 3
2. Rollings, A. (2020). mlrose: Machine Learning, Randomized Optimization and SEarch package for Python, hiive extended remix. https://github.com/hiive/mlrose. Accessed: 27 Sept 2020.
3. Hayes, G. (2019). mlrose: Machine Learning, Randomized Optimization and SEarch package for Python. https://github.com/gkhayes/mlrose. Accessed: 27 Sept 2020.

4. Kingma, Diederik, and Jimmy Ba. (2014) Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.