CS7641: Analysis and Report for Assignment 2- Random Optimization

Mingming Zhang

mzhang607@gatech.edu

All experiments in this assignment were conducted using ***mlrose*** module in python.

Four peaks optimization problem

The four peaks fitness function is defined as the equation shown below:

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

Where:

- $tail(0,x)$ is the number of trailing b's in x;
- $head(1,x)$ is the 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.

In the experiment that I conduct, the four peaks problem that I define is with 100 data points, and the threshold parameter that I used was 0.15. Thus, the threshold value will be calculated as 0.15*100 = 15. Ideally, we can have the first 84 values to be 1, and the last 16 values to be 0. Thus, we can earn 84 points from the number of head, and 100 points because both head and tail are bigger than the threshold. So, the maximum fitness value that we can get is 184.

I am interested in this problem because it is a straightforward example with two local optima that is with wide basin, which is a good example to show how some of the algorithm could be trapped in the local optima and how others may escape it to find a better solution. This problem is with two local optima, each of them is with a wide base. The fitness function will also have two sharp global optima at the edges of the space. As a result, this problem design would likely to cause random hill climbing (RHC) and simulated annealing (SA) algorithm to be trapped in one of those two local optima, as they are with a very large basin. The reason for that is RHC are comparing its direct neighbors to decide which direction to move next, while SA is select a point from the nearby neighbors of the current given point to decide the next step move. The design of these two algorithms is likely to cause them trapped in local optima like in the four-peak problem. We can see the results shown in Table 1 that both RHC and SA algorithms are reached at a level of fitness value of 100, which is likely of one of the local optima. This experiment was running with different trials with different parameters, and both RHC and SA algorithms at most reached at 100 in fitness value several times.

Genetic algorithm (GA), on the other hand, is able to achieve its max value at 166, which is much better than either RHC or SA algorithms. I think the reason is that GA algorithm is based on using cross over to mutate the previous fittest individuals and replace the least fit individuals in the population. This working process not depending on the data points in the neighbors and thus more likely to jump outside the local optima to explore new examples. In contrast, MIMIC algorithm is with the worst performance. I think the reason could be that MIMIC algorithm is designed to solve dependency tree like problem and does well in complex structure, while the four-peak problem is more like a chain-structure, which might not be suitable to use MIMIC algorithm to solve.

Table 1: performance comparison of different algorithms when solving Four Peaks problem

Algorithm	Total running time (s)	Total running iterations	Best fitness values
Random Hill Climbing	2.024	31565	100
Simulated Annealing	6.016	75396	100
Genetic Algorithm	1491.201	50583	166
MIMIC	2538.199	457	21

One thing to notice that is the time cost. RHC and SA algorithm is running with high number of iterations, but each iteration of these two algorithms cost little to compute. Thus, these two algorithms spend about 2s and 6s, respectively, to finish all the computation with the parameters that was set in this experiment. To reach a much better performance, GA algorithm has to try a lot of attempts during each search to find a better state. Thus, this process cause GA algorithm to spend a much longer time, about 1491s, to find this ultimate better solution. Here I want to point out that GA is finding its maximum fitness value after about 550 iterations. So, we could potentially stop searching at earlier iterations after reaching its maximum value to save a lot of

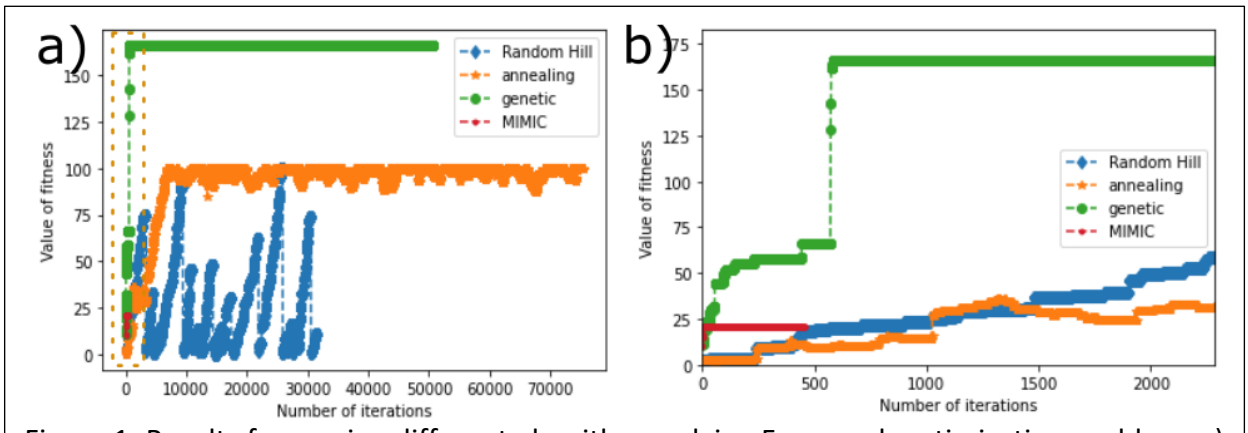

Figure 1: Results from using different algorithms solving Four-peak optimization problem. a) the fitness curve as the total number of iterations during the optimization process. b) the zoom-in of the dashed rectangular box highlighted part of the figure shown in a).

computational time. Also, GA algorithm didn't find the theoretical max value 186, but with its found maximum fitness at 166, which could explain that GA algorithm is probably stuck in a better local optima, but not find the global optimal solution.

The other algorithm that is very time consuming is MIMIC. Because MIMIC algorithm is trying to represent probability distribution for each data point and designed to solve dependency tree like problem. Thu, it is very time consuming to calculate this information during each iteration. Although, it takes less iteration to find a solution, but the total time cost is the longest among all the algorithms in this experiment.

In conclusion, GA algorithm is the best algorithm to use here to solve this four-peak problem, but with a high cost in computation time.

Max K-color optimization problem

Fitness function for Max-k color optimization problem is defined as following. Given a n-dimensional state vector $x = [x_0, x_1, \dots x_{n-1}]$, Where x_i denotes the color of node i. The problem evaluates the fitness of this n-dimensional vector, as the number of pairs of adjacent nodes of the same color.

After several rounds of experiments with different parameters, I choose to set the "restart" parameter to be 5 with 100 maximum attempts during each search when using RHC algorithm. When using SA algorithm, I set the maximum attempts to be at 200 with its initial temperature set at 200, minimum temperature set at 1 with decay rate of 0.85. The above-mentioned setting with these two algorithms can both achieve the maximum value 2237 in this experiment. However, When I looked at the time expenses for these two algorithms, simulated annealing is much faster with time cost of 0.7567 when finishing solving the problem (Table 2). I think one

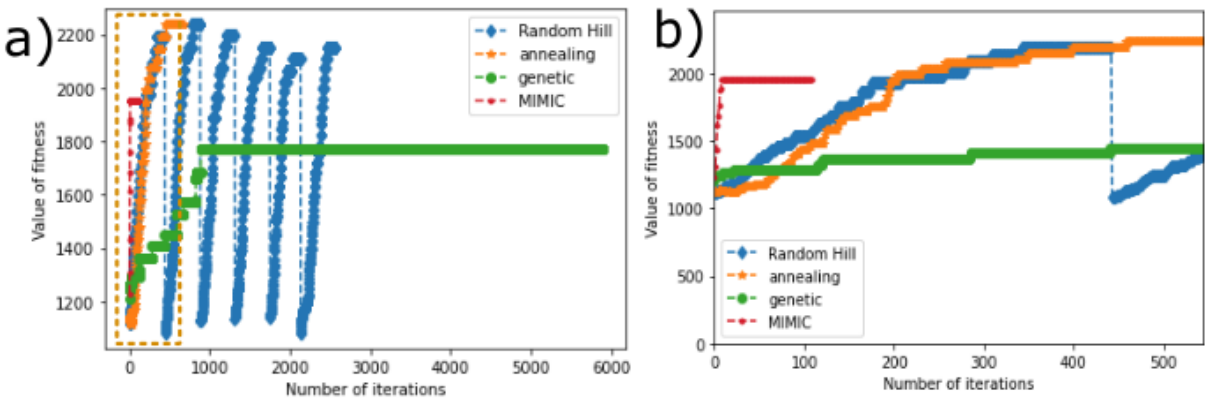

Figure 2: Results from using different algorithms solving K-color optimization problem. a) the fitness curve as the total number of iterations during the optimization process. b) the zoom-in of the dashed rectangular box highlighted part of the figure shown in a).

reason to explain this is the SA algorithm is designed in a way so that it can explore much more neighbors when searching for best fitness value. It also has higher temperature at the beginning to allow a larger exploit area, with a smaller temperature to allow its fine tuning of the search. It contains the search power of RHC but has larger capacity and refine searching strategy. One the other hand, RHC is easy to be stuck in a local optimum and the restarting of search helps this problem but with the expense of spend longer time.

Table 2: performance comparison of different algorithms when solving K-color problem			
Algorithm	**Total running time (s)**	**Total running iterations**	**Best fitness values**
Random Hill Climbing	3.168	2560	2237
Simulated Annealing	0.7567	660	2237
Genetic Algorithm	1850.399	5889	1772
MIMIC	608.514	109	1952

In this optimization problem, GA and MIMIC algorithm both can not find a better solution compared SA and RHC. While the amount of time GA and MIMIC algorithm needs is much higher. Because both GA and MIMIC are based on the design of changing an initial population of samples. It is just GA is using mutation and MIMIC is using the fittest examples to estimate the probability distribution for next iteration. Thus, these two algorithms usually take much longer time to solve a given optimization problem. While the complexity of this Max-K color problem is not very high, so the more complex algorithm like GA and MIMIC cannot solve it very well with a reasonably short period of time.

In conclusion, based on the max fitness value each algorithm can find and the time expenses, I think SA algorithm is better when solving the max K-color optimization problem.

Knapsack optimization

The knapsack problem is defined as following:
Given a set of n items, where i has know weight w_i and known value v_i. Given maximum capacity value w. The Knapsack fitness function evaluates the fitness of a state vector $x = [x_0, x_1, \ldots\ x_{n-1}]$, and it is fitness function is defined as following:

$$Fitness(x) = \sum_{i=0}^{n-1} v_i x_i, \text{ if} \sum_{i=0}^{n-1} w_i x_i \leq W, \text{ and } 0, \text{ otherwise,}$$

Where x_i denotes the number of copies of item i include in the knapsack.
I found this problem interesting because it is very relevant to the practical problem when are facing in our daily life. For example, if you are general manager of the sports team. Each player has their own salary contract while you need to keep the total salary you spend for this team under the cap of the league. In another scenario, we have multiple projects running in a company

and we want to spend as much money as possible to maximum resources for each project, but also need to control the total money spent under a limit amount of budget.

When running the experiment with RHC, I set the algorithm to restart for 20 times with 4000 attempts within each search. When using SA algorithm, I increase the max attempts in each search to be 10000. From Figure 3 we can tell that, RHC is with the worst performance, while SA was able to perform significantly better to find a solution that is close to the best solution (Table 3, Fig.3).

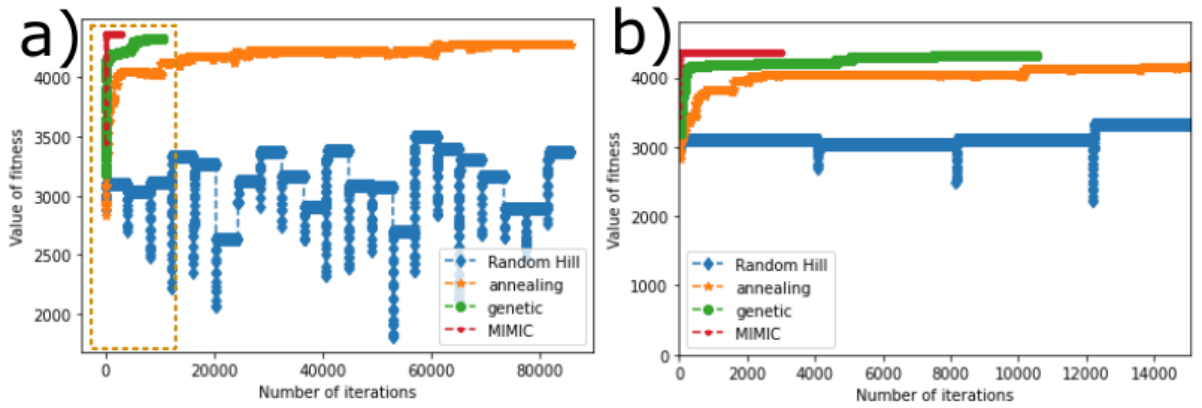

Figure 3: Results from using different algorithms solving Knapsack optimization problem. a) the fitness curve as the total number of iterations during the optimization process. b) the zoom-in of the dashed rectangular box highlighted part of the figure shown in a).

Table 3: performance comparison of different algorithms when solving Knapsack problem

Algorithm	Total running time (s)	Total running iterations	Best fitness values
Random Hill Climbing	7.662	85497	3496
Simulated Annealing	10.408	85477	4273
Genetic Algorithm	229.078	10505	4328
MIMIC	16427.6288	3015	4362

When using GA and MIMIC algorithm to solve this problem, I use 3000 attempts during each search with a population size of 200. From the results shown in Fig.3 and Table 3, MIMIC can find the best solution among all the algorithms used here. The k-napsack problem is a harder problem with no polynomial time solution. The strength of MIMIC algorithm is to exploit the underline structure of the problem space and can get more information. Thus, it has better capability of solving more complex problem, but solving the more complex structure isn't free but with high computational expenses. Thus, even MIMIC uses the least number of iterations, it takes the most time to finish solving this problem (Table. 3). In addition, GA and SA algorithms are also have similar capability in terms solving this problem, because their finding of best fitness values is just slightly lower than MIMIC algorithm (Fig.3 and Table 3). RHC has least performance level in solving this problem.

One thing to pay attention is amount of time used when solving this problem. MIMIC takes about 16428s to finish solving this problem, while SA and GA use about 10.4s and 229s, respectively, to solve this problem. If we are allowed with rich computation resources, using MIMIC algorithm will achieve the best results. But if we are limited by the amount of resources that we have when solving this problem in real life, choose GA or SA algorithm could be the selection and the downside maybe to sacrifice the performance a little bit.

Using random optimization algorithms to learn neural network parameters

In this section, I investigate using RHC, SA, GA to find the weights for a pre-defined neural network with 3 layers and each layer with 30, 50, 30 hidden nodes respectively. Initially, we use gradient descent search conduct the first experiment to get a baseline performance when solving this weights optimization problem. We conduct all the experiments with 2500 iterations, 10000 attempts within each iteration, unless otherwise stated. The data I use from Assignment 1 is MINIST dataset, with 10000 training examples and 4898 testing examples. The original data is from here http://yann.lecun.com/exdb/mnist/. In this dataset, each example is an image of a handwritten digit with its true labeled class from 0 to 9, which is corresponding the handwritten digit in the image. In *mlrose* module, the loss is simply defined as the negative value of the fitness.

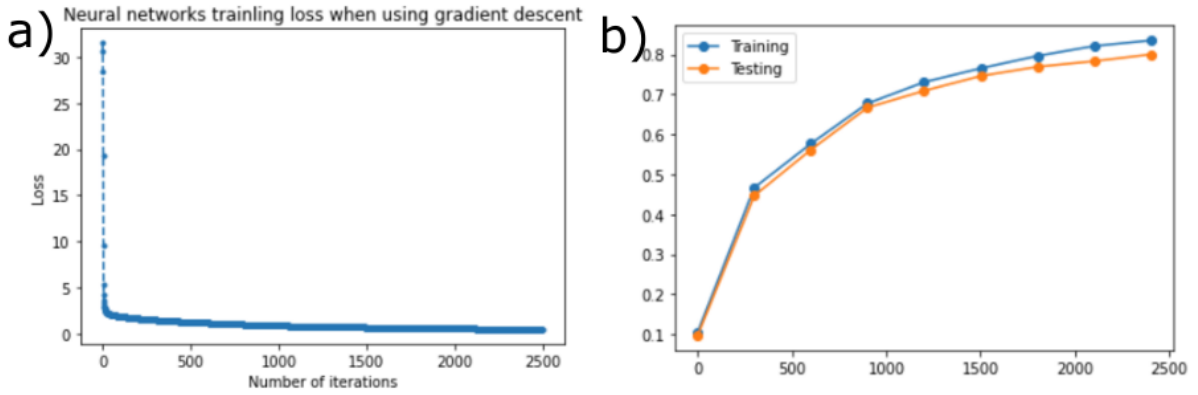

Figure 4: The performance as the number of iterations when using gradient descent algorithm to solve the optimization weights for neural network. a) loss during training as the number of iterations. b) training and testing accuracy when using the same amount of data to conduct training using different number of iterations.

I first use the *mlrose* module to repeat the experiment with gradient descent (GD) method to optimize the weights. After 2500 iterations, the training accuracy can reach about 84% and

testing accuracy can reach about 80% (Fig.4). This result is in a comparable range when using *scikit-learn* module on this dataset. From the accuracy curve shown in Fig4b, if we continue to increase the number of iterations, we may can increase the accuracy even more. However, the goal here is to compare the performance of different algorithms, so I didn't try to maximize the performance when using GD algorithm. Table 4 shows the overall performance comparison when using GD, RHC, SA and GA algorithms.

Table4: performance comparison when solving the optimizing weights for the neural network

	Gradient descent	Random hill climbing	Simulated annealing	Genetic algorithm
Training accuracy	0.84	0.0505	0.08562	0.2456
Testing accuracy	0.80	0.052	0.0795	0.2485
Training time (s)	687.75	1797.53	544.412	7239.877

Overall, we can see that none of the algorithms I tried can achieve similar performance compared to using gradient descend algorithm. The fact that the other three algorithm perform so much worse. Neural network's loss function (fitness) is with very complicate surface, even with three layers of hidden nodes (30, 50, 30) shown in this experiment. Random hill climbing is so easily to be trapped in one of the local optima when conducting the search, because there are so many of these local optima. As a result, no matter how many restarts for the search we do, it helps little about the performance. Although, we can see that the loss is improving as the number of iterations increase, the total amount of loss changes is only about 0.0025 after 2500 iterations (Fig.5). From the results when using gradient descent algorithm (Fig.4), this improvement in loss is very tiny. As the fitness function is with very complicated shape, it is very likely that the RHC is always trapped in the same local optima, no matter how many iterations it was running. So, we can see when using RHC, training and testing accuracy is consistently very low (Fig.5). In addition, running RHC to solving such complex problem with very high number of iterations and attempts is taking a long time. When finishing running with RHC algorithm, the total amount of time spent was 1797s, which is much longer than using GD algorithm (Table 4).

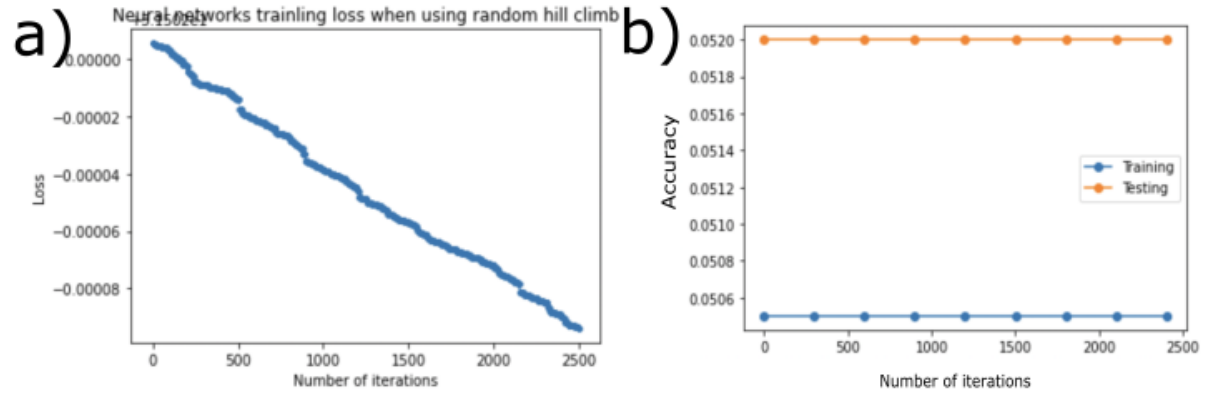

Figure 5: The performance as the number of iterations when using random hill climbing to solve the optimization weights for neural network. a) loss during training as the number of iterations. b) training and testing accuracy when using the same amount of data to conduct training using different number of iterations.

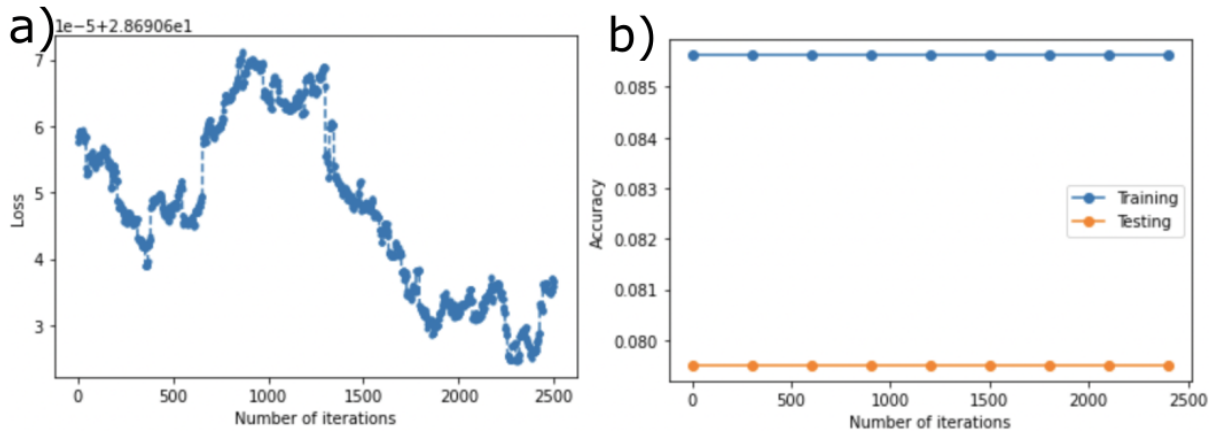

Figure 6: The performance as the number of iterations when using simulated annealing to solve the optimization weights for neural network. a) loss during training as the number of iterations. b) training and testing accuracy when using the same amount of data to conduct training using different number of iterations.

Next, I change the algorithm to be SA in solving the neural network weights optimization problem. When using SA algorithm, I set the initial temperature to be 500, with a decay rate to be 0.85 with minimum temperature to be 4. As we can see the amount of improvement in loss by the end of 2500 iterations maybe be bigger than when using RHC algorithm (Fig. 6). However, the improvement is not big enough to make the optimized weights to perform well when decoding with the testing data with training accuracy at 0.086 and testing accuracy at 0.079 (Fig.6). When we look at the design of SA algorithm, it works like an improved version of RHC, with the search area to be several neighbors of the given data points. However, SA algorithm is still easily to be

trapped in some local optima as the design of this type of algorithm. Although it's better and faster when using SA than using RHC, it is probably trapped in another local optima which is better than RHC but still a long way from the global optima solution.

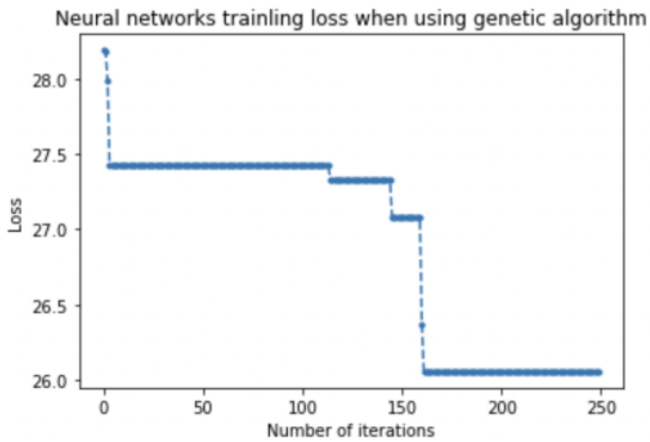

Figure 7: The performance as the number of iterations when using genetic algorithm to solve the optimization weights for neural network. The figure shows the loss during training as the number of iterations.

When I switch to using GA algorithm, I see some improvements of the algorithm performance as increasing the number of iterations (Fig.7). Here, please pay attention to that the number of iterations was set to 250 and the total number of attempts was set to 500 due to the very high expense in computation when solving the weight optimization problem for neural network. From Table 4 above, we can see that even with shorter number of iterations and attempt, it still takes about 7240 s to finish solving this problem. Compared to GD, SA, RHC, this is a much longer period. So, I didn't recompute this problem using different number of iterations when given that same training and testing data. The predicting accuracy was much improved with 24.6% and 24.9% on training and testing data, respectively, compared to when using RHC and SA algorithm. However, this performance is still way lower compared to the baseline experiment with over 80% of accuracy when using GD to solve this problem. One possible improvement in the future we can do is to increase the number of iterations and attempts within each iteration to see if that can help improve the performance.

From the above results, it is clear that RHC, SA and GA algorithms are not very capable in solving the defined neural network weight-optimization problem with an architecture of 3 layers of neurons. Neural network decision boundary with such architecture should be with very complex surface, where none of RHC, SA and GA algorithm can solve very well. Most likely, these three algorithms were stuck in some local optima when conducting the search. Although, GA algorithm is better compared to SA and RHC, but probably was stuck in a better local optimum.