CS7641 Machine Learning Assignment 2: Randomized Optimization

He Yao

hyao66@gatech.edu

PART 1. INTRODUCTION

The purpose of this assignment is to explore different randomized optimization algorithms and to compare their performances when applying to different problems. The four randomized optimization algorithms explored are: RHC, SA, GA and MIMIC.

Random Hill Climbing (RHC): Hill Climbing is a method that move towards the neighbors that has higher fitness value, and eventually reach the local optima. Randomized Hill Climbing applies random restart so that it has better chances to "escape" from the local optima and being able to find the global optima.

Simulated Annealing (SA): Simulated Annealing is an analogy from how to forge wrought iron in the old days, when the blacksmith increases the temperature to smooth the edges of the metal and decreases the temperature to hold the structure. Simulated Annealing applies the similar idea and try to find the balance between exploring and exploiting. With high temperature, the algorithm is doing exploring by randomly seeking different points, similar idea as random restart for RHC. While temperature cooling down, the algorithm is exploiting nearby neighbors to find the one with maximum fitness value. With repeating this heating and cooling process, SA should eventually reach the global optima.

Genetic Algorithm (GA): Professor explained Genetic Algorithm by an analogy of biology that offspring is produced by interactively mating and mutating different parts to get the best result and eliminate the 'bad' parts. I think this analogy is very intuitive and help me understand the algorithm. For each iteration, each element in the population is the outcome of this 'crossover', and is weighted by its relative fitness value versus the fitness value of the whole population. One of the disadvantages of GA is that the search space can grows exponentially with number of parameters/elements increases.

Mutual Information Maximizing Input Clustering (MIMIC): Different from the previous three algorithms, MIMIC solves the optimization problem by estimating the probability density function and sample from it for each iteration. Hence, MIMIC can 'memorize' previous iterations and find the better probability density function from the structure through iterations, and find the optima value. Although it takes less iterations, each iteration takes longer time compare with other algorithms since it need to estimate probability density function.

PART 2. APPLY RANDOMIZED OPTIMIZATION TO 3 PROBLEMS

Three optimization problems are Four peaks, Max K color and Knapsack. I applied the above 4 optimization algorithms (RHC, SA, GA, and MIMIC) to these 3 problems and discuss the advantages and disadvantages of each algorithm when apply to each of these 3 problems.

Four Peaks (GA is the best to solve for this problem): This problem is defined on bit strings with length 100. And the parameterized value t_percent is set to 0.1 (T = 10), with reward 100. The fitness function is maximized if the string with 100 elements can get reward of 100 and at the same time the number of tails with 0 and number of heads with 1 as big as possible. It seems that we can find 2 local optima and 2 global optima of the 4 peaks problem in this setting. With T = 10, 189 is the global optima, with two ways to reach 189. 89 leading ones with 11 ending zeros, or 11 leading ones with 89 ending zeros. There are two local optima, with either all ones or all zeros for 100 elements, which will lead to function value of 100.

We can see from the below figure 1 and table 1, RHC quickly find the local optima of 100 and stuck in the local optima even with random restart, with significant large amount of iterations (about 45K). This is because the RHC is difficult to make 'correct decision' when searching for large plateaus, and the basins to trap the RHC with local optima is quite large compare with the basin to find the global optima. SA performs well and able to find the global optima with very short amount of time even with significant iterations (9.7K). I set the heating and cooling schedule quite aggressively by setting the initial temperature of 100 with geometrically decaying schedule with decay factor of 0.8. It seems with enough of iterations, SA is able to find the global optima with 100 elements. This is because SA is a good method to search for large search space with relatively short amount of time. However, SA is also vulnerable to stuck at local optima if we have "bad luck" with bad initial state. GA performs well and able to find the global optima as well. By design, GA should work well on the four-peaks problem, since the algorithm should keep the 'good parts' of the parent states and mutate the 'bad parts' to produce children states. With about 16K iterations, GA converged to the global optima. MIMIC does not seem to be good solution for four-peaks problem, which is expected since learning previous states cannot help much on finding better current state. Although MIMIC requires significant less number of iterations (here only 112 iterations), estimating the probability density function for each iteration greatly increase the run time.

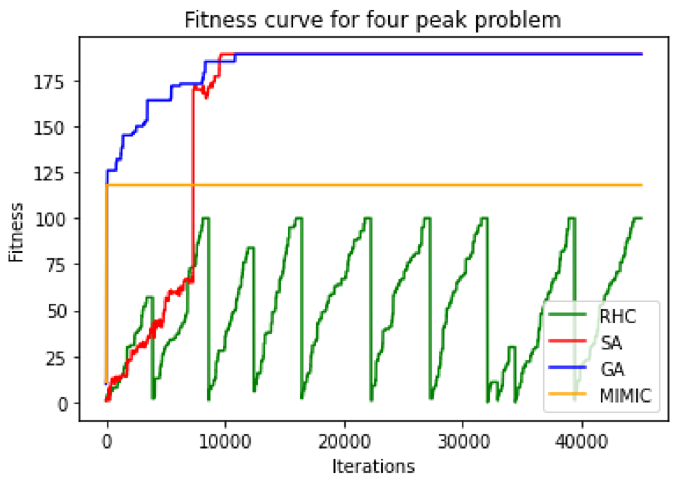

Figure 1 — Fitness curve – four peaks

Algorithm	**Time (second)**	**Iterations**
RHC	1.66	44896
SA	0.54	9747
GA	346.49	15832
MIMIC	642.97	112

Table 1 — Performance – four peaks

When tunning hyper-parameters for GA, there are three parameters to tune with no limits on iterations. I would like the algorithm to determine when to stop when it thinks it reaches the optima. I explored population size: [100, 200, 500], max_attemps: [100, 500, 1000, 5000] and mutation probability: [0.001, 0.005, 0.01, 0.05, 0.1]. Several observations listed below:

- Larger population on average, better than smaller population
- Larger max attempts in general, better than smaller max attempts
- Smaller mutation probability works better for four-peak problem

With all the combinations explored, population size 200 or 500, with mutation probability of 0.001 and max attempts of 5000, GA can find the global optima.

<u>Max K Color (SA is the best to solve for this problem):</u> mlrose define the max k color problem to maximize the number of adjacent nodes that have the same color. Here I use 100 nodes, and randomly generate 3000 edges from these 100 nodes to be connected. It is not difficult to think through this problem that the maximum fitness value is actually the number of edges (in my case 3000) as long as no node is isolated.

We can see from below figure 2 that RHC, SA and GA all achieve the maximum value of 3000, while MIMIC did not achieve maximum value. It is expected for RHC and SA to achieve maximum value since this problem does not have local optima to trap these two algorithm, and as long as we allow RHC and SA to look at sufficient enough number of neighbors (in mlrose, parameter max_attemps). From below Figure 1 we can see that RHC find the maximum after second restart. So if I increase the number of max_attempts for RHC, it should be able to find the maximum at the first step, I don't even need it to restart. Although MIMIC take smallest number of iterations, it does not find the maximum value in my experiment. And each iteration take significant longer time compare with other algorithms, since estimating probability density function takes time.

With least amount of time, small number of iterations, and achieve maximum value, SA is the best algorithm for Max K color problem.

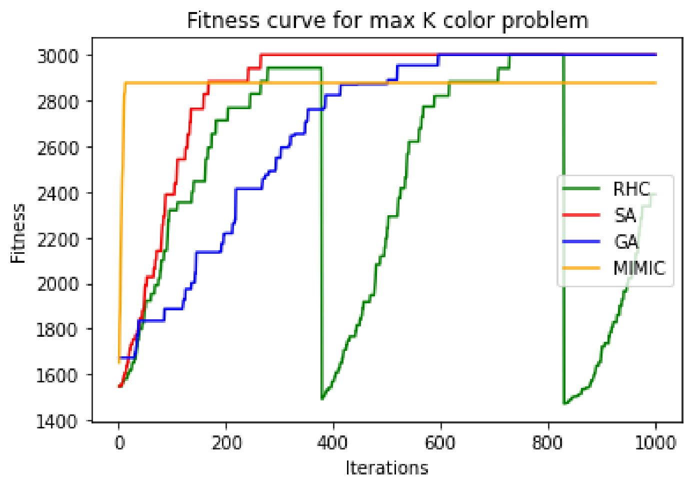

Figure 2 — Fitness curve – Max K color

Algorithm	Time (second)	Iterations
RHC	65.65	43791

Algorithm	Time (second)	Iterations
SA	0.57	365
GA	191.65	696
MIMIC	658.13	113

Table 2 — Performance – Max K color

Knapsack (MIMIC is the best to solve this problem): Knapsack is an optimization problem that given a set of items, with predefined weights for each item and value for each item, to optimize the total value of items to be included in a collection but subject to weight limit. We would like to maximize the value of items in the collection. For this assignment I am solving a 0-1 knapsack problem with 100 items, which restricted the number of copies of each kind of item to 0 or 1, and weight limit set to 0.6 of the total weight. Knapsack is an interesting problem with real-world applications, e.g. asset management when selecting investing portfolios. Knapsack problem is also NP-hard (non-deterministic polynomial-time hardness), which means it could not be solved in polynomial time without approximation.

From the results of the four algorithms shows in below Figure 3 and Table 3, we can see that MIMIC is the best algorithm for this problem, with the ability to find the solution way better than other 3 algorithms with much less iterations. This is expected since with more complicated problem like knapsack, MIMIC is able to learn the underlying structure of the problem through iterations and provide a solution closer to optima and more reliable.

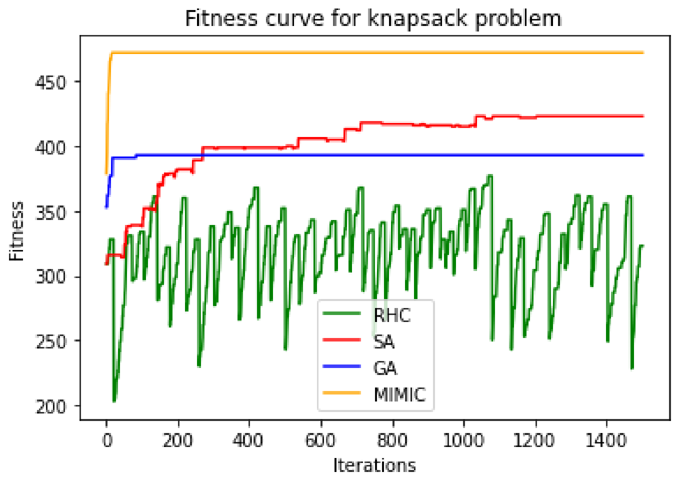

Figure 3 — Fitness Curve – Knapsack

Algorithm	Time (second)	Iterations	Max value achieved
RHC	3.16	34597	408
SA	0.15	1304	423
GA	5.11	184	393
MIMIC	636.22	117	472

Table 3 — Performance – Knapsack

To summarize, RHC and SA are suitable for problems that are simpler in structure, and experience learned from previous iterations are not very important, and the search space for local optima is not significantly bigger than the space for global optima. GA and MIMIC are suitable for problems with complex structure, and previous experiences matters. In terms of computation time cost, GA and MIMIC require more time to run even with less iterations.

PART 3. APPLY RANDOMIZED OPTIMIZATION TO NN

The dataset used in this section is the customer segmentation dataset used in assignment1, which has over 8000 instances and 9 independent variables describing the features of the customer including gender, education level, spending score etc. The dependent variable is the segmentation of these customers with 4 categories. This problem is quite difficult and the NN trained with gradient descent in assignment 1 can only reach accuracy of a little above 50%.

The purpose of this section is to compare the performance of NN with gradient descent with randomized optimization methods. Below figure 4 replicate the performance of NN with ***gradient descent*** using mlrose package, to serve as our base case to compare. I set the iterations to 2500, we can see from the learning curve that the accuracy is very difficult to get improvement when training accuracy reach about 50%.

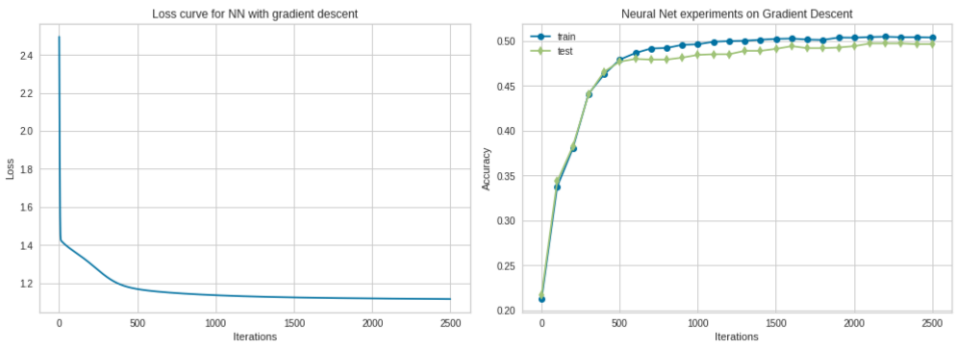

Figure 4— NN with gradient descent, loss curve (left) and learning curve(right)

Using ***random hill climbing*** to find the weight for NN instead of gradient descent, the training accuracy can only reach 27%, while the testing accuracy can only reach about 25%. Increasing number of iterations and/or more complex NN structure with more layers does not significantly help with the accuracy. Please refer to below Figure 5 for loss curve and learning curve.

We can see that the improvement of fitness curve is really very limited, from 2.5 to 2.0 over 2500 iterations, which consistent with the observations that increasing number of iterations does significantly help with the accuracy. With a complex problem like the customer segmentation data, we would expect the search space has very complex structure and a lot of local optima. Hence RHC is very easy to trap in one of the local optima and cannot perform well. We can see that the training curve and testing curve has a constant gap that these two curves cannot converge, which also suggests that the algorithms is trapped at local optima and hence cannot generalize well on the testing data.

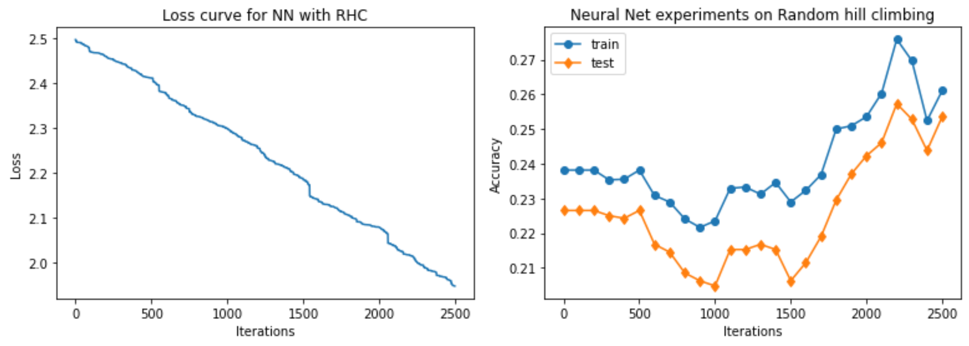

Figure 5— NN with randomized hill climbing, loss curve (left) and learning curve(right)

Using ***simulated annealing*** to find weight for NN, we face similar situation as random hill climbing that the methods is trapped at local optima and increasing iterations does not significantly help with the accuracy. The best accuracy we can reach with SA is about 27% for training. Both RHC and SA is sensitive to initial start point with the customer segmentation data, with different random state, the resulting accuracy can be different. This is because we may get 'lucky' to trapped at better local optima with good initiation point to start of search.

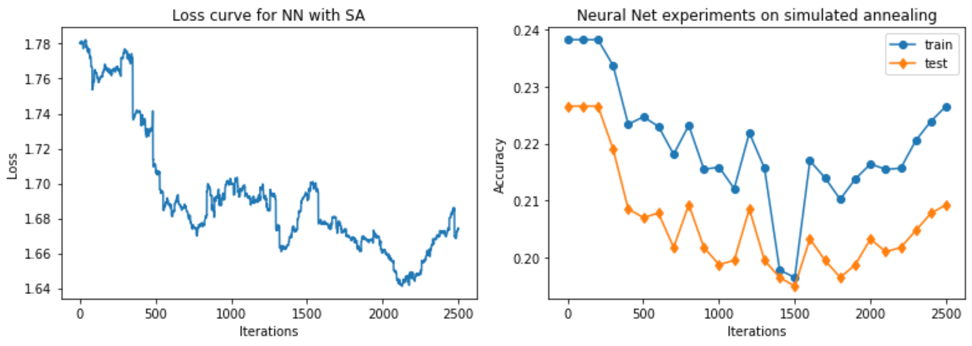

Figure 6— NN with simulated annealing, loss curve (left) and learning curve(right)

There are several hyper-parameters to tune for ***genetic algorithm***, max attempts for each iteration, population size and mutation probability. Since training NN with genetic algorithm takes tremendous amount of time, I only use 5 iterations to tune these hyperparameter. I explored population size: [100, 200, 500], max_attempts: [100, 500, 1000, 5000] and mutation probability: [0.001, 0.005, 0.01, 0.05, 0.1]. With population size = 500, mutation probability 0.001, and max attempt 100 is sufficient with increase max attempt does not help with accuracy, even with 5 iterations, test accuracy can reach 32% which is much better than RHC or SA.

One interesting observation is that GA reached peak testing accuracy of 32% at third iterations with population size 500, mutation probability 0.001, and the accuracy keep the same with increasing iterations till 2500 as I tested. And also the performance of NN with GA is very sensitive to the initial random point, with choosing different random state greatly affect the accuracy, sometime can as worse as only 22% testing accuracy. And GA takes tremendous time to train, with 2500 iterations takes about 6 hours, testing beyond 2500 makes computationally prohibitive. But in general, increasing population size improve the accuracy, while mutation probability needs to be carefully selected based on different problem.

Below table summarize the testing accuracy and training time over 2500 iterations for Customer Segmentation data for NN using Gradient Descent, Random hill climbing, Simulated Annealing and Genetic Algorithm. Overall, no algorithms can beat our benchmark, Gradient Descent. I think this observation make sense since given complex problem with a large search space with many local optima, naïve search algorithm can easily get trapped, and cannot find the best direction with the steepest step to the optima. Gradient descent is able to compute the best direction along which we should change our weight that is mathematically guaranteed to be the direction of the steepest descent.

Algorithm	**Time (second)**	**Testing Accuracy**
Gradient Descent	645	49.96%
Random Hill Climbing	313	25.36%
Simulated Annealing	390	25.95%
Genetic Algorithm	19829	32.26%

Table 4 — Performance Neural Net