CS7641 Machine Learning:
Assignment 2 Randomized Optimization

Judy Jungeun Cha
jcha64@gatech.edu

1 INTRODUCTION

The purpose of this report is to explore randomized optimization. The random search includes to implement algorithms to three optimization problems such as Travelling Salesman Problem, N queens, Flip Flop Problems. The local random search algorithms include Randomized Hill Climbing (RHC), Simulated Annealing (SA), a Genetic Algorithm (GA) and Mutual Information Maximizing Input Clustering (MIMIC). After this step, the first three algorithms are applied to find good weights for a neural network instead of using backprop which I used in previous assignment. My wine data was to test which is white or red wine with features information like alcohol, sugar level and pH.

2 OPTIMIZATION PROBLEMS

2.1 Travelling Salesman Problem (TSP)

Travelling Salesman Problem asks the following questions: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?" We need to travel every city and come back to start city given that we can visit each node(city) only once. It is an as hard as the NP problem in computational complexity theory. [1]

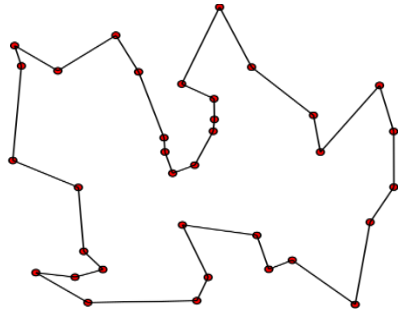

Figure 1. Solution of a TSP: the black line shows the shortest possible loop that connects every red dot. from Wikipedia [1]

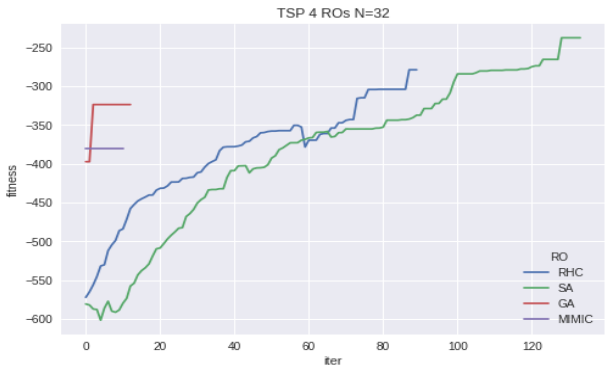

Figure 2. Fitness vs. Iterations (TSP), N=32 (32 cities)

Based on above graph, GA is the most optimal solution for TSP. GA converged very quickly with only few iterations (less than 5 iteration) and reached the best fitness score of about -320. Within the same iteration period, RHC and SA are far less than GA and

MIMIC stopped at the level of -380 and does not improve anymore.

Figure 3. Wall Clock Time vs. Size N (TSP)

GA takes long time to run because it determines fitness for everyone in the population and performs crossover/mutation. However, GA reaches the optimal with only a few iterations in our set up (N=32), it is much faster than MIMIC with a half of time (5 vs. 15). SA is the fastest, but GA's fitness score is much higher than SA in ~ 90 iterations.

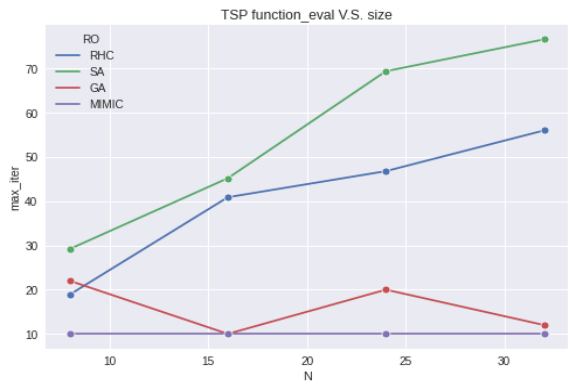

Figure 4. Max Iteration vs. N (TSP)

In addition, if we look at the max iteration vs. N(cities), RHC and SA requires much more iterations with upward trend as N increases, but GA needs very small iterations almost as low as MIMIC does (N=32). The number of iterations for MIMIC is not correlated to size N at all.

GA searches population of individuals like a parallel random restarts. When there are multiple local optima, GA is very effective to escape local optima. Especially the number of parameters (large N) is large, GA has advantage since we noticed GA does not require more iterations proportionally even though running time is relatively high.

2.2 N Queens (NQ)

When n = 8, NQ requires placing eight queens on 8 * 8 chess board to avoid an attack of eight queens given that queens can move horizontally and diagonally. No queens can be in the same of row, columns, positive diagonal, and negative diagonal. The eight queens' problem has 92 distinct solutions (64! / 56! * 8!).

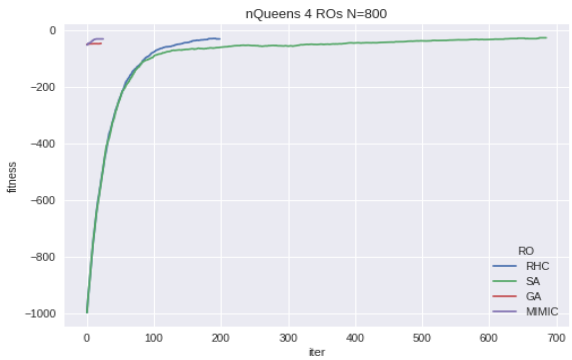

Figure 5. Fitness vs. Size (nQueens), N=800

In terms of fitness score results, GA and MIMIC reached the optimal point with a few iterations in Fitness vs. Size graph.

Considering execution time, GA performs the best for nQueens.

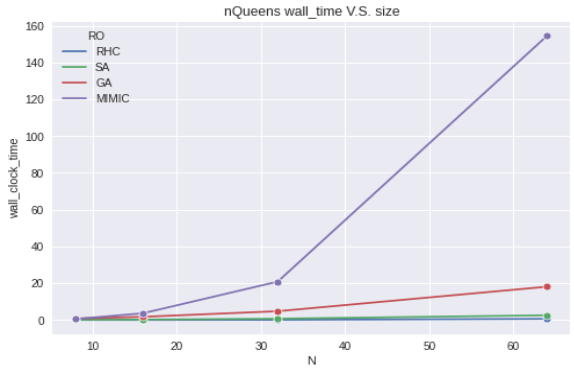

Figure 6. Wall Clock Time vs. Size (nQueens)

For nQueens, SA requires more iteration to converge compared to other algorithms, but SA is the one of the fastest algorithms with RHC. In addition, even though N (size) increases a lot, wall clock time does not change at all for RHC and SA. In this sense, SA is useful to very large optimization problems like airline scheduling. I want to highlight the advantages of SA here.

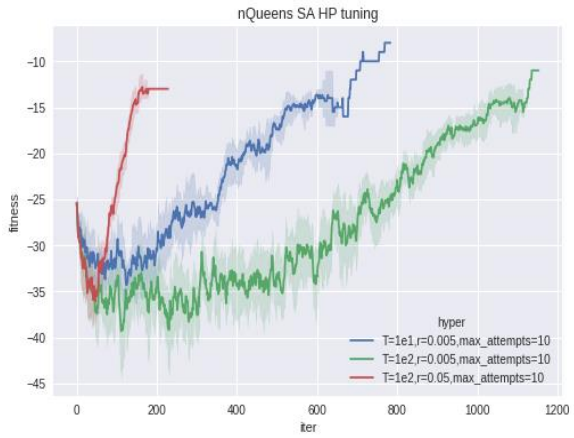

Figure 7. SA Hyperparameters Tuning (nQueens)

SA is very useful when there are a lot of local optima where gradient decent would be stuck often. SA starts at a random point and jump to new point based on the probability function P (x, x_t, T).

$T(t) = \max(T_o - rt, T_{min})$

Based on HP tuning graph, SA performs the best when T_o is higher and r is lower (T=1e2, r=0.005). Here T_o is initial temperature at time t=0 and r is temperature decay parameter. When T is big, the possibility of any move to new point ($e^o = 1$) is getting big but if T is small, the possibility of moving to new point ($e^{infinite} = 0$) is low and it performs only uphill to improve fitness.

The blue line in the graph proves higher T_o and lower r (decay rate) performs better than the others. It reaches higher fitness function than the red line and it takes less iterations than the green line. We start with high T at the beginning for more exploration and decrease T slowly to reach the best configuration. SA largely depends on the parameter tuning like shown in the graph.

2.3 **Flip Flop (FFP)**

The Flip Flop problem look for the total number of times of bits alternation within a bit string. Given that N (bit count) is 800, fitness values results for 4 algorithms are as follows:

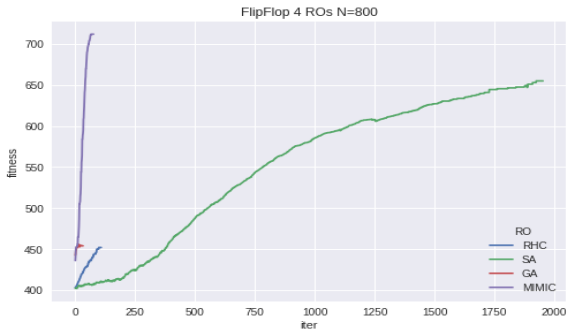

Figure 8. Fitness vs. Size (FFP), N=800

In terms of fitness values, MIMIC performs the best with over 700. Next SA will reach 660 at 2,000 iterations.

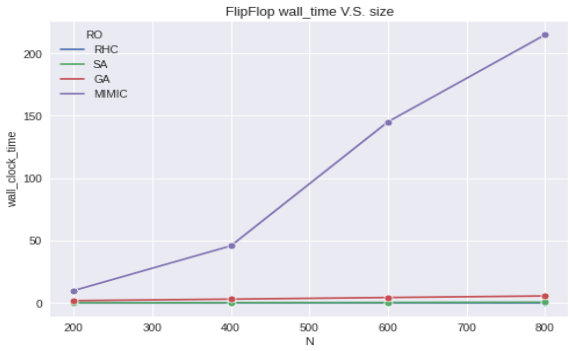

Figure 9. Wall Clock Time vs. Size (FFP)

However, as expected, MIMIC takes more than 200 times time compared to other algorithms when N=800. If we consider execution time as well, SA will be the most effective algorithms in FFP. However, even though there is no change of required iterations for other algorithms, SA significantly requires more iterations as N (bit count) grows. But SA still be the best algorithms for FFP as low execution time and high fitness score.

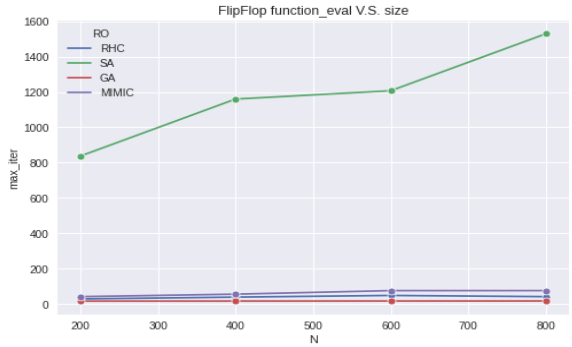

Figure 10. Max Iteration vs. N (FFP)

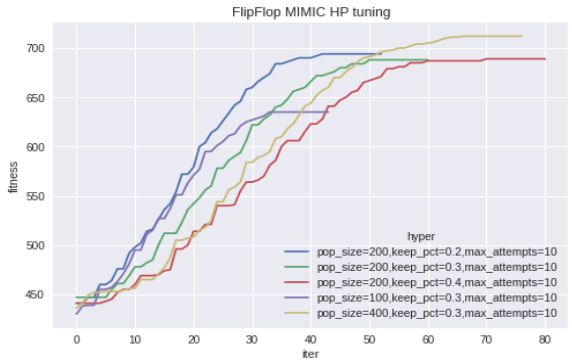

Figure 11. MIMIC HP Tuning (FFP)

For flip flop problem, MIMIC performs the best when population size=200, keep_pct=0.2 which is blue line in the MIMIC HP tuning graph. Even though the yellow line produces higher fitness score after 50 iterations, the blue line performs the best if we consider execution time too. The keep_pct (float, default: 0.2) is proportion of samples to keep at each iteration of the algorithm, expressed as a value between 0 and 1. [2] Lower proportion of samples to keep at iteration works better.

While other 3 algorithms only consider points and are unclear about probability distribution, MIMIC conveys the data structure and directly model the distribution and successively refine the model. Accordingly, MIMIC does well with structure as a probability model. As we observed, MIMIC is very slow because it does a lot of work in every iteration to estimate probability distribution.

2.4 **Summary**

Here is my summary based on implementation of 4 algorithms to three optimization problems. Based on complexity and specifications, it is important to select the right algorithms for each problem.

Algorithm	Advantage	Disadvantage
RHC	- Fast - Not expensive (constant factor)	- Not good for complication problem
GA	- Good at local search - Good at complicate problem	- Slow - Need to tune many parameters
SA	- Sometimes fast - Sometimes good at large size - Good at many local optima	- Slow - Need to tune many parameters
MIMIC	- Works well with structure - Directly model distribution and refine model	- Very slow - Need to tune many parameters

Table 1: RHC, GA, SA, and MIMIC comparison

3 NEURAL NETWORK WITH OPRIMIZATION

Neural network uses gradient decent (G-D) to find good weights for backpropagation process. Here we will try randomized optimization algorithms in finding optimal weight instead of backprop. Wine data has binary feature (white wine:1, red wine: 0) as response variable. In the previous assignment, I tested networks of nodes with the different number hidden layers with 'tanh' and 'relu' activation function and the best result was 5 of neurons and three hidden layer size with tanh activation function. Here I used 5 of neurons and 1 hidden layer since the results of ((5,5,5) 'tanh') was very similar to ((5), 'tanh') and ((5,5), 'tanh') in assignment #1.

3.1 RHC

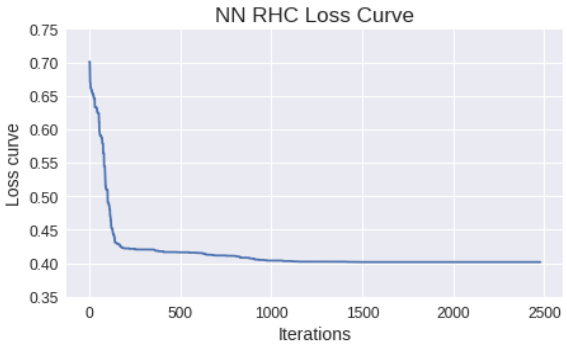

Figure 12. Loss Curve (RHC)

RHC's loss curve dropped sharply within 100 iteration and reached the global optima quickly. RHC runs fast to optimal result as the fastest algorithms compared to others for my wine data. For wine data, RHC fitness function is not that expensive to compute as there are not many neighbors. Three features (sugar, alcohol, and pH) in wine dataset have not highly correlated each other. It helps RHC run faster. Also, RCH's accuracy is the highest as well. RHC gives the most optimal solution for wine data.

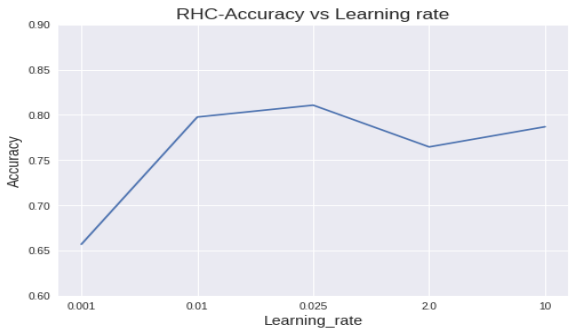

Figure 13. Accuracy vs. Learning rate (RHC)

For running rate, it increases sharply up to 0.01 and reached the maximum accuracy of 0.81 at 0.025 of learning rate. After 0.025, it maintains between 0.75 and 0.8 accuracy. This is very different from G-D accuracy vs. learning rate graph in below. For G-D, accuracy drops sharply after passing maximum learning point of 0.001. In G-D, learning rate over 0.001 does not help much for accuracy.

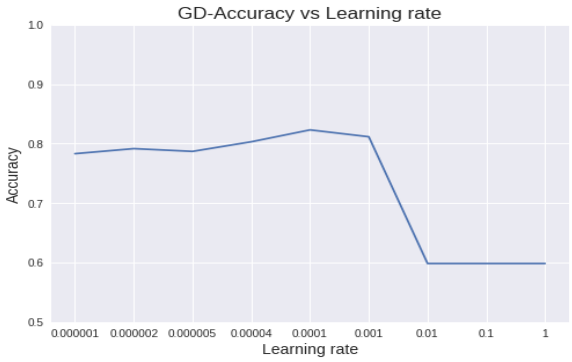

Figure 14. Accuracy vs. Learning rate (G-DC)

3.2 GA

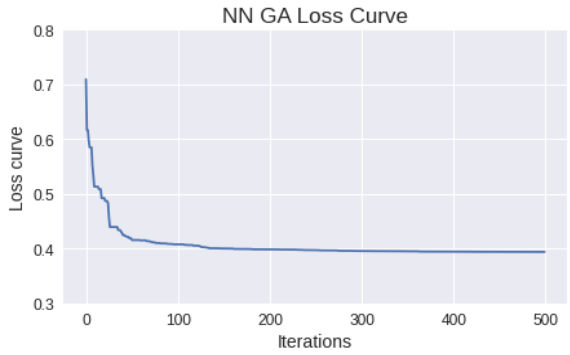

Figure 15. Loss Curve (GA)

Like RHC, GA's loss curve reached at the global optima quickly around 100 iterations. However, GA's one iteration takes much longer than RHC's iteration because GA first computes fitness function for all and then selects most fit individuals and replaces the least fit individuals by crossover/mutation. GA performs the worst in terms of wall clock time.

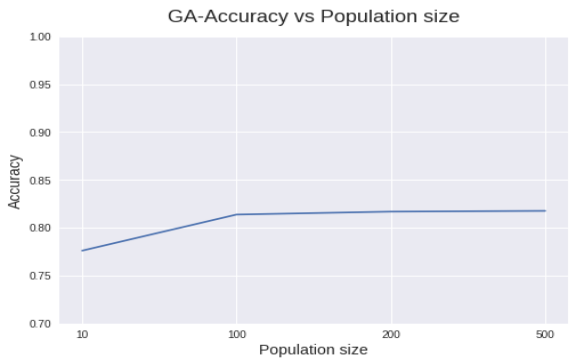

Figure 16. Accuracy vs. Population Size (GA)

Since GA takes time the most but the time does not increase much with increase of size. Given that the population size is [10, 100, 200, 500], GA reaches the maximum accuracy at population of 100 and maintains after. It seems wine data set is simple enough that after 100 tries with crossover/mutation, there is not many things left to improve afterwards.

3.3 SA

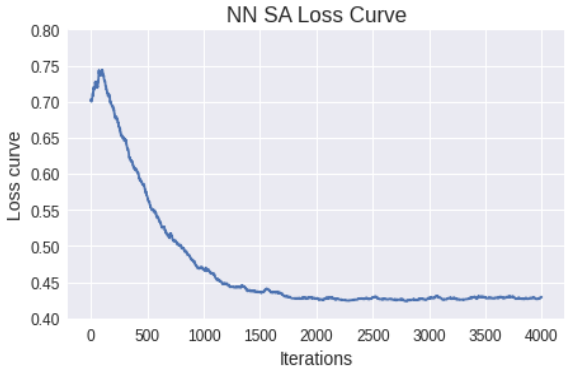

Figure 17. Loss Curve (SA)

Unlike other algorithms, SA loss curve increases at first and decreases slowly. It is because SA searches samples by jumping to new samples with given probability at the beginning and decreases temperature T slowly. With this exploring time, SA is the second slowest algorithms after GA, but it is still a lot faster than GA. SA is converged slowly after 1,000 iterations. It requires more iterations to converge compared to other algorithms which takes less than 100 iterations.

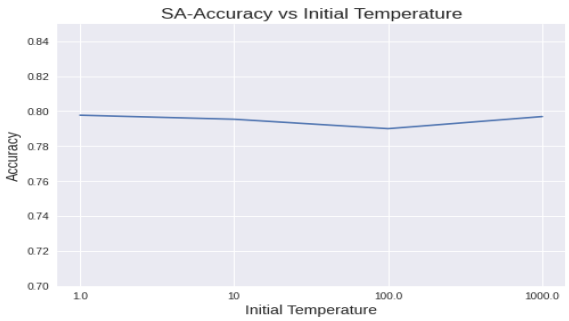

Figure 18. Accuracy vs. Initial Temperature (SA)

One of parameters of SA is initial temperature. Based on above SA accuracy vs. initial temperature graph, initial temperature parameter does not affect much for wine data. I tried 1, 10, 100, 1000 initial temperature but accuracy does not change much over the initial temperature changes.

3.4 Gradient Decent (G-D)

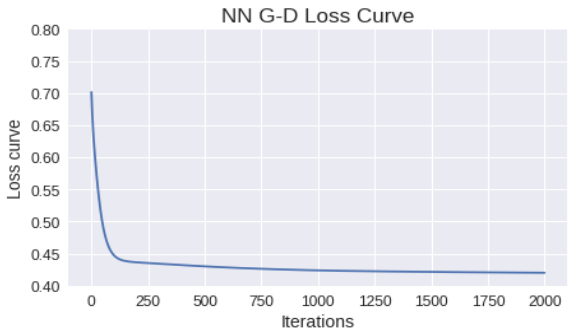

Figure 19. Loss Curve (G-D)

G-D (backpropagation) is the benchmark for randomized optimization algorithms. As expected, G-D converges quickly around 100 iterations. Since we tested only training sample, it does not increase again later as the loss curve of validation set would show due to overfitting.

3.5 Summary

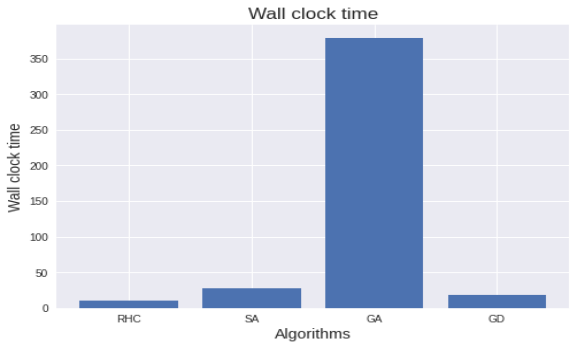

Figure 20. Wall Clock Time for All 4 Algorithms

Obviously, GA takes wall clock time the most. Wine data is relatively simple with three variables, but it has almost 7,000 samples. Since GA's running time is somewhat related to sample size, 7,000 samples take time to compute. Next, SA takes time due to searching time and RHC is the fastest since the fitness function of wine data is not that expensive with relatively simple data structure with only three variables of sugar, pH and alcohol.

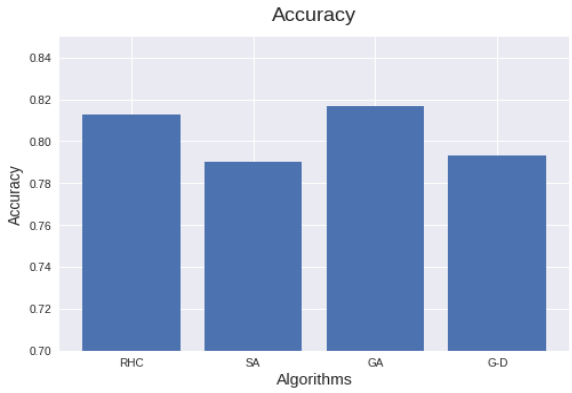

Figure 21. Accuracy Results for All 4 Algorithms

For wine dataset, RHC and GA give best accuracy results and it finds optimal weights for neural network. But since GA takes the

longest time among algorithms, RHC gives the best results in terms of execution time and test accuracy for wine data. It is because wine data has relatively simple data structure and fitness function is not expensive to compute for RHC, but 7,000 samples size takes time to compute for GA.

References:

1. https://en.wikipedia.org/wiki/Travelling_salesman_problem

2. https://readthedocs.org/projects/mlrose/downloads/pdf/stable/