Randomized Optimization

Spring 2021 CS7641 Assignment 2

Hung-Hsi Lin   hlin83@gatech.edu

ABSTRCT

This paper aims on applying random search algorithms including randomize hill climbing, simulated annealing, genetic algorithm and MIMIC on three different optimization problem domains. Besides, the first three algorithms shown above will be implemented in Neural Network for analyzing real dataset. The performance of each algorithm in different conditions will be analyzed.

INTRODUCTION

The whole paper could be divided into 2 main parts. First, three different optimization problems domains: N-Queen, Knapsack and Continuous Peaks will be solved by randomize hill climbing, simulated annealing, genetic algorithm and MIMIC algorithms separately. In the second part, some algorithms are used to replace the back-propagation algorithm in Neural Networks for determining the weight distribution. The prediction performance accuracy will be calculated for each algorithm and compared. First, the theories of four randomize optimization algorithm should be briefly introduced:

Random Hill Climbing (RHC)

The algorithm starts from selecting a random point in the domain, then perform hill climbing [1]. The point checks its neighbors' fitness values and move there if the value is larger than itself. This algorithm runs iteratively until a min/max point is converged and reached. Sometimes RHC gets stuck in a local min/max, and it could be possibly solved in the next iteration with random starting point.

Simulated Annealing (SA)

The name of the algorithm comes from material science while scientist repeatedly cool and heat materials to remove the defects in the crystal [2]. There's also a temperature parameter in this algorithm. As the temperature is high, the point has higher possibility to go downhill, thus explore more region even though the reward is lower. On the other hand, as the temperature is low, the point tends to only go uphill. The algorithm starts with high temperature, so there's more probability to explore different regions at beginning. As the iteration is larger, temperature decays and the selections become more conservative. In SA algorithm, the high temperature prevents the situation from getting stuck at local peaks.

Genetic Algorithm (GA)

The idea of this algorithm comes from gene evolution. A group of candidates are generated first for solving a problem, the candidate has higher fitness value tends to survive and contribute to a part of the next generation of candidates. Mutation of the candidate also contributes to evolution, explore more possibility for higher fitness [3]. At the end, the candidates will converge to the same solution.

MIMIC

The full name of MIMIC is Mutual Information Maximizing Input Clustering. This algorithm constructs structures for estimating solutions in each iteration. Not like above three algorithms, only the optimized points were saved, the MIMIC saves the function and refines structures iteratively, and at the end, it provides the optimized structures for estimating solution [4].

SECTION 1

Three optimization problems will be solved by four algorithms and the corresponding advantage/disadvantages will be analyzed in this section.

1.1 N Queens

The N Queens problem is to search for a state that all queens won't attack with other queens, if it doesn't attack, then we add points. Therefore, it is a problem searching for maximum fitness scores. First, I defined a customized function to determine the fitness, then create a problem using mlrose_DiscreteOpt method since the value of states will be discrete integer numbers [5]. Then I implemented some functions to tune and optimize the proper parameters in each algorithm for this problem. The optimizing progresses are concluded in Figure 1. For RHC, the parameter of restarts, which is used to determine numbers of random restarts is optimized. In Figure 1.a, restarts of 4 is required for the case of max attemps is 55, while max attemps means the maximum number of states to find a better state. On the other hand, while the max attemps is only 10 in Figure 1.b, larger

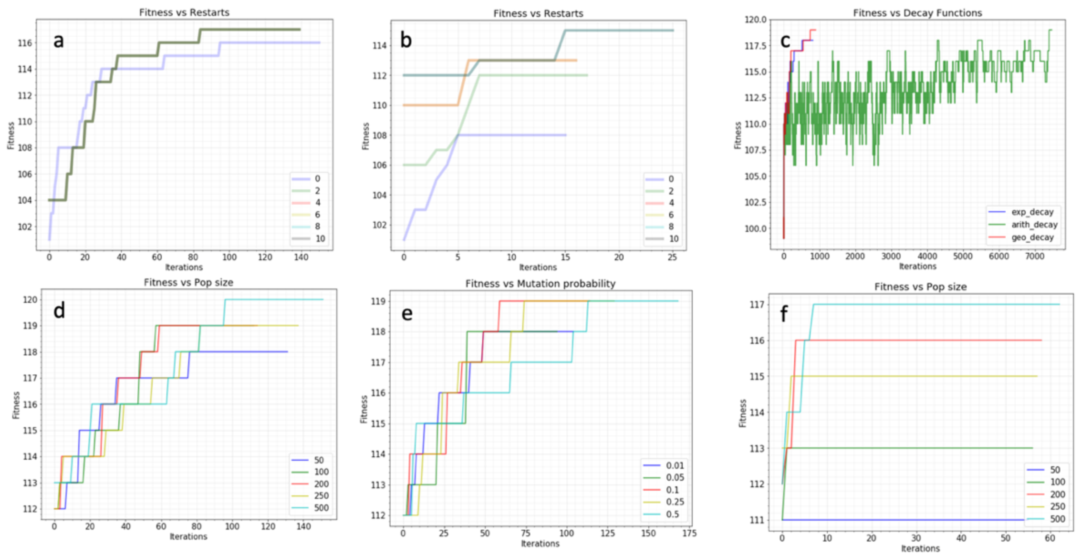

Figure 1 – Fitness scores under variant iterations for (a) RHC with restarts as variable for max_attemp = 55, (b) RHC with restarts as variable for max_attemp = 10, (c) SA with different decay function, (d) GA wit pop size as variable, (e) GA with mutation probability as variable, (f) MIMIC, with pop size as variable.

restarts number (8) is required to make the solution converge to the best fitness score. For SA, the type of decay is an important parameter, and in Figure c we could clearly see the arithmetic decay takes a lot of iterations to converge compared to exponential and geometric decay, demonstrating the fact that arithmetically reducing the temperature is too slow and inefficient. For GA, the size of popped candidates for sure is important, and in Figure 1.d we could see that if the size is less, then the fitness might be bad because the variance of the model is small and not generalized enough. However, if the size is too large, then it takes more iterations to converge (cyan line with pop size of 500). Mutation probability is an important parameter for GA. In Figure 1.e, larger possibility of mutation takes more iteration to converge, and small mutation prob results in worse fitness score and smaller variance. For MIMIC, the pop size is also important, in Figure 1.f, we could see if the pop size is too small, the convergence happens faster but with large bias, same idea as the case for GA.

After finely tune the parameters in each algorithm for N-Queen problems, then I implemented them into 8, 32 and 64 queen problems and record the fitness score, iteration and time for each algorithm, which are summarized in Figure 2. As we could see clearly from Figure 2. a to c, GA (red line) worked pretty well in both less required iterations and higher fitness score. In GA, the processes of combination and mutation helps the algorithm solves complex questions. We can imagine that the state vector is divided into two part, and the algorithm tends to select the good candidates to survive into next generation, and the required change will be less and less, so the scores will keep increasing until converged. On the other hand, the RHC and SA takes a lot more iterations to converge because the question is too complex for them, simply checking neighbors is not enough for this problem. From Figure 2. d to f, we could see the time required for each algorithm to find the best solution. Even though the time required for GA is more than SA and RHC, the better score and less iterations still makes **GA** as the best algorithm for solving the N-Queen problem.

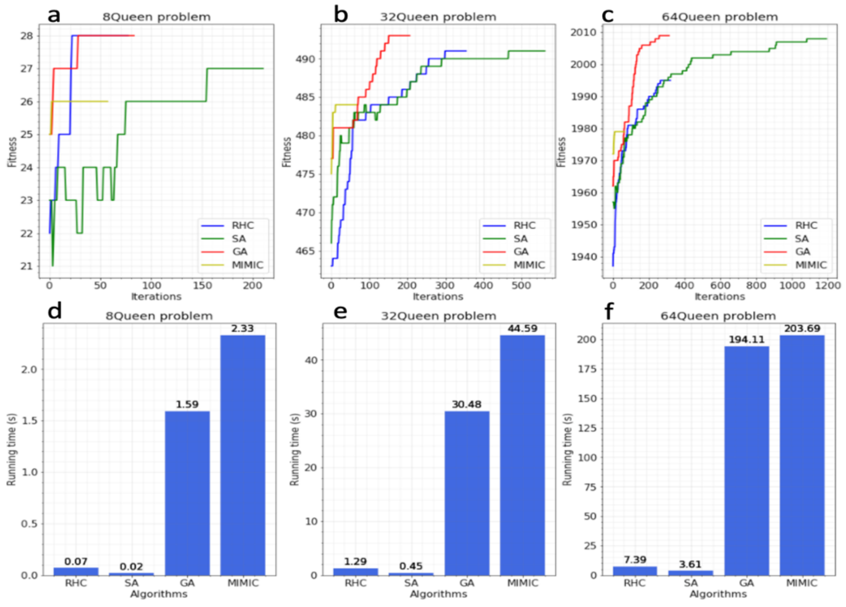

Figure 2 – Fitness scores under variant iterations for (a) 8 Queen, (b) 32 Queen, (c) 64 Queen problems. Required time to converge to the best solution is concluded in (d) 8 Queen, (e) 32 Queen, (f) 64 Queen problem.

1.2 Continuous Peaks

The continuous peak is a similar problem to 4 and 6-peaks. I directly built the problem by importing the ContinuousPeaks class in mlrose module [5]. The 0's and 1's could be anywhere in this problem. Therefore, we need to search for the longest (global max) 0 or 1 string and avoid some sub-long (local max) cases. Same procedures as before, I first implement the same parameter tuning scripts to determine the best settings for each algorithm in this problem then applied in different dimensions of continuous peak problem (10, 30, 200), the results are summarized in Figure 3.

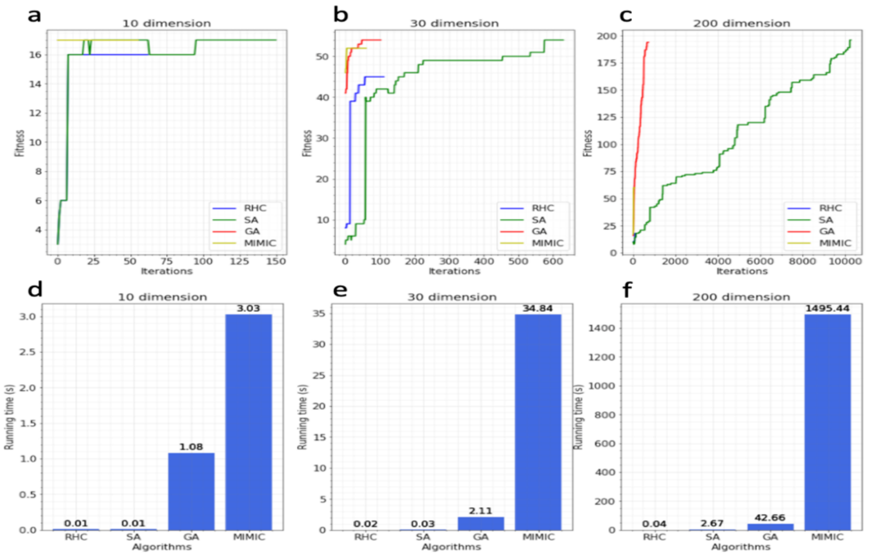

Figure 3 – Fitness scores under variant iterations for (a) 10, (b) 30, (c) 200 dimension of continuous peaks problems. Required time to converge to the best solution is concluded in (d) 10, (e) 30, (f) 200 dimension of continuous peaks problem.

From Figure 3.a to c, the fitness of RHC is low because there are multiple local max/min in continuous peaks problem, demonstrating the fact that RHC is easy to get stuck in these local peaks if the number of restarts is not large enough. In terms of fitness score, MIMIC, SA and GA are all able to converge roughly at the best fitness result, while the required iterations for SA is way larger than GA and MIMIC in the problems with higher dimensions. The required time for solving continuous peaks problems is shown in Figure 3.d to f, and it can be seen that the time required for MIMIC is crazily high compare to the other three algorithms. The time GA needs is also ~ 20 to 100 times more than SA. Given the fact that both SA and GA provide accurate answers, and the total time for SA is shorter than GA, **SA** is the best among these algorithms for solving continuous peak problem. The reason of SA takes a lot of iterations but with shorted required time is that the processing time in each iteration is shorter for SA than GA and

MIMIC. In conclusion, there are a lot of local peak in this problem, where RHC is easy to get stuck. This problem is not as complex as N-Queens which needs time-consuming algorithms like GIMIC and GA., therefore SA is the best option which has capability to avoid getting stuck at local peaks and complete the optimization correctly in a relatively short runtime.

1.3 Knapsack problem

For Knapsack problem, at the beginning a fixed size of objects which have assigned weights and values are generated. The goal is to put the objects into a knapsack without overweight, and at the same time holding the highest summed values. I first generated random weights and values with length of size. Then use the Knapsack class in mlrose module to create the problem [5]. The returned state will be a list of state with 0 or 1 to represent if the object is selected or no in the knapsack, and this is a problem to search for maximum values.

Again, the procedure starts with finely tuning the parameters on each algorithm, and the optimized algorithms for knapsack problem are then implemented into the knapsack problem with the size of 10, 20 and 50. The fitness and runtime results are concluded in Figure 4.

From Figure 4. a to c, we clearly see that the problem is too complex for RHC and SA. The fitness scores from these two algorithms are roughly only 80% compared with GA and MIMIC. As the size of bag is bigger, the difference is getting larger as well. The performance of MIMIC and GA is pretty close, but the time for MIMIC is longer than GA for bigger size as we could see in Figure 4. d to f. Knapsack is a complex problem which needs to keep the good objects among all the candidate, so that these good choices could survive into the next iteration. SA and RHC can't deal with the problem at this level of complexity. GA has the property that there is higher probability in keeping the good genes into the next generation of candidates. MIMIC also tends to keep better structure for retain successful states. Therefore, **MIMIC** and **GA** are both good randomize algorithm for knapsack problem. The performance of MIMIC is slightly higher than GA, therefore, for this problem, if the time is a concern, GA is better, on the other hand, for better fitness, MIMIC is the good choice.

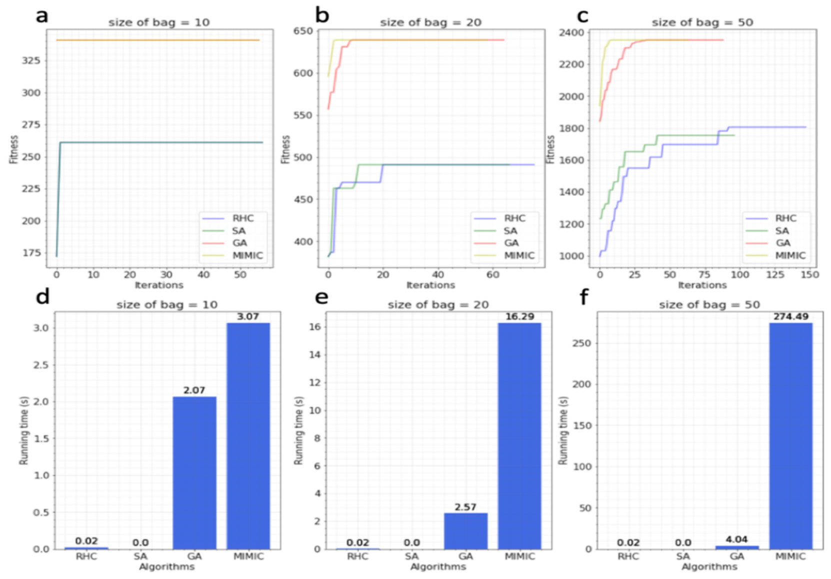

Figure 4 – Fitness scores under variant iterations for (a) 10, (b) 20, (c) 50 bags of knapsack problems. Required time to converge to the best solution is concluded in (d) 10, (e) 20, (f) 50 bags of knapsack problems.

2 NEURAL NETWORK OPTIMIZATION

In this section, instead of using backprop function on neural network model, SA, RHC and GA are implemented for calculating weight distributions. The dataset is the same as assignment 1, diabetes.csv which has 15000 instances, 10 attributes with continuous attributes such as age, BMI, and discrete output, 0 or 1 to represent if the patient has no or has diabetes. Before implementing the algorithm, the dataset was undergone preprocessing including dropping NA, normalization and rescale. Then the data was split into training set (80%) for training/validation and test set (20%) for determining the final f1 scores.

2.1 RHC

Before testing the performance of RHC algorithm on diabetes dataset, the NNGS-Runner was used to run the grid search to determine the most optimized parameters for RHC in this problem. The learning rate, activation function and hidden layer size were determined to be 0.1, relu function and [5, 10], respectively. the f1 score was calculated for test and training sets, with cross-validation of 5 to avoid overfitting issues. In Figure 5.a, I tried to input different numbers of restart into RHC algorithm and calculate the test error. The test errors for all case reduce dramatically within the iteration between 10 to 500, then gradually converge. The error for restarts of 9 has lower error because it avoids the situations of getting stuck at local peaks, which happens more often as the restarts is low. In Figure 5.b, different maximum attempts were tested, and we could clearly see as the attempts is set too low, 5 in our case (red line), the RHC converges too fast and gets stuck at local peaks instead of global min/max, resulting in large errors even with high iterations. The test and train error are summarized in Figure 5.c using RHC, two curves match with each other pretty well and train error is slightly slower as expected, demonstrate the fact that the built model using RHC has low variance for this dataset.

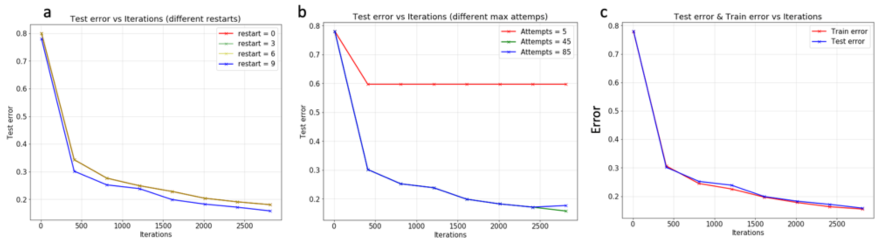

Figure 5 – (a) Test errors vs iteration with number of restarts = 0, 3, 6, 9.

(b) Test errors vs iterations with different max attempts of 5, 45 and 85.

(c) Test and training error vs iteration.

2.2 SA

The setting of Neural Network including the learning rate, activation function and hidden layer size are as same as the case of RHC so that we could make fair comparison between randomize optimization algorithms. In Figure 6.a, the test error between different decay factors for geomDecay function in SA are applied into Neural Networks. It is obvious that the decay rate of 0.99 is different from others, which converges slower. The reason is that as the decay factor is high, the point has higher possibility to explore the region with lower fitness score at low iterations, therefore, for the first 1000 iterations, the error from fitness of decay=0.99 is larger than the other cases. In Figure 6.b, same story as RHC case, the max attempts should be high enough, in our case 45, to avoid converge too early which results in large error. In Figure 6.c, the trend is similar to the case of RHC, the trends of two curves are similar and train error is slight lower than test errors. The performance between RCH, SA and GA will be compared in later section.

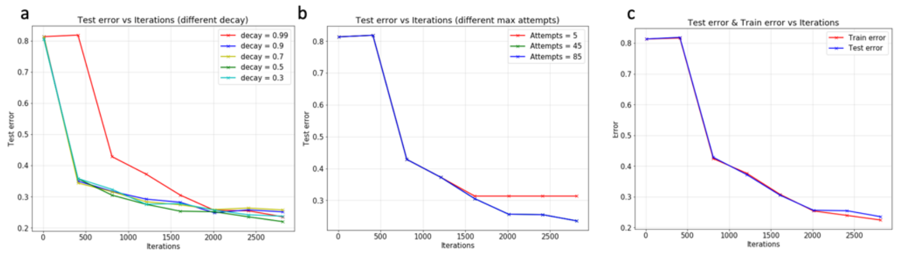

Figure 6 – (a) Test errors vs iteration with variant decay rate. (b) Test errors vs iterations with different max attempts of 5, 45 and 85. (c) Test and training error vs iteration for SA algorithm.

2.3 GA

The Neural networks settings for GA is as same as previous two algorithms. In Figure 7.a, the error for GA is large even the mutation prob was tuned, the same trend was also found for pop size. Therefore, in Figure 7.b, I take the hidden layer as variables and it could be seen that the hidden layer and nodes influences the performance of GA algorithm a lot. If the hidden layer size is too small, the error could be as large as 0.47 after converging. GA needs more hidden layers/nodes for building low bias Neural Networks than SA and RHC algorithm. One interesting thing we could see from the plot is that for all cases the result gets converge within 500 iterations, supporting the fact that GA needs less iterations for finding the best fitness results. In Figure 7.c, the test and train error are shown, the trivial difference demonstrates the facts that the Neural Networks models trained by GA has low variance.

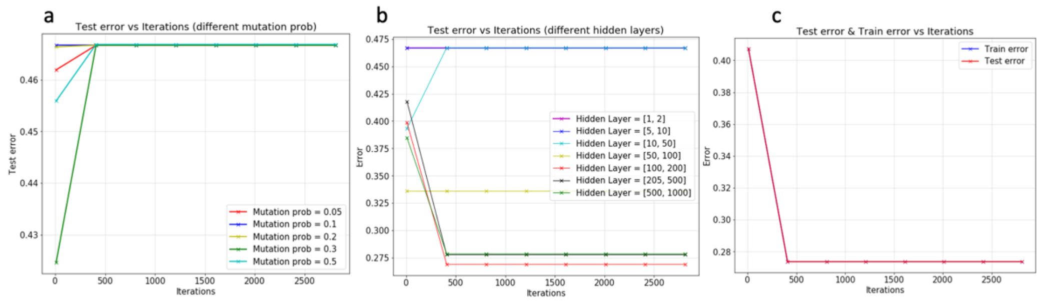

Figure 7 – (a) Test errors vs iteration with variant decay rate. (b) Test errors vs iterations with different max attempts of 5, 45 and 85. (c) Test and training error vs iteration for SA algorithm.

3 SUMMARY

The performances on predicting the diabetes dataset in Neural Networks using different optimization algorithms are summarized in Figure 8.a, and the corresponding time for training is shown in Figure 8.b. Gradient descent (GD) provides the best testing accuracy and it is the most efficient algorithm. The performance of RHC and SA are close, SA needs more iterations at beginning due to high temperature, and the time for RHC is slightly more than SA because a larger number of restarts (8) was set in this test. GA needs way more time than others and the performance is the worst among these algorithms. Even though the required iteration to converge is lower than others, the total time is still high because the processes in each iteration is expensive. These observations demonstrate the fact that as using Neural Network in this dataset, the weight distribution is a relatively smooth curve, not too many local peaks would cause the optimization to be stuck. Therefore, SA and RHC outperformed the GA in this case, and even non-advanced GD could easily calculate proper weight distribution for minimizing testing errors. As the dataset is more complex, with more hidden layers/nodes, we expect that GA could have a better performance.

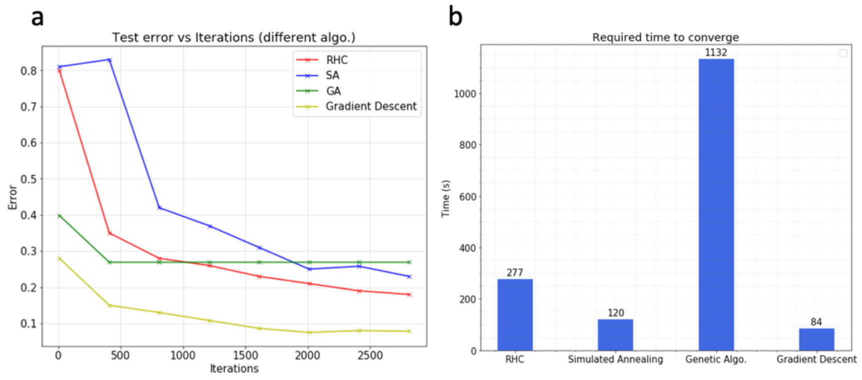

Figure 8 – (a) Test errors vs iteration executed by 4 algorithms and (b) corresponding required time to converge.

4 REFERENCES

1. https://en.wikibooks.org/wiki/Artificial_Intelligence/Search/Iterative_Improvement/Hill_Climbing
2. http://web.mit.edu/dbertsim/www/papers/Optimization/Simulated%20annealing.pdf
3. https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_introduction.htm#:~:text=Genetic%20Algorithm%20(GA)%20is%20a,take%20a%20lifetime%20to%20solve.
4. https://ieeexplore.ieee.org/document/6098783
5. https://readthedocs.org/projects/mlrose/downloads/pdf/stable/