CS 7641 Machine Learning Assignment 2

Randomized Optimization

Tao Peng

1. Introduction

This assignment is to explore and study randomized optimization. We applied four different randomized optimization algorithms, Randomized Hill-Climbing (RHC), Simulated Anealing (SA), Genetic Algorithm (GA) and Mutual-Information-Maximizing Input Clustering (MIMIC) to four different problems: Neural Network, Four Peaks, Traveling Salesman, and Knapsack. We observed and analyzed the different behaviors of those algorithms in different circumstances. After doing this assignment, I gained a much deeper understanding of these randomized optimization algorithms. I used ABAGAIL [1] for all the algorithms, and used or modified some Java codes to run the algorithms and get the results from them.

2. Problems

2.1 Neural Network Problem

This problem is to find the best weights for a neural network. In Assignment I, a neural network was for calssifying the data sets. We used back propagation there to find the best weights. However, here we are going to use randomized optimazation algorithms instead, and compare the behavior with back propagation, as well as compare among different randomized optimazation algorithms themselves. For the data set, I decided to use the Wine Quality Data Set [2] from Assignment I so that I can directly compare with the previous results.

The Wine Quality Data Set is to determine the sensory quality of the Portuguese “Vinho Verde” wine. There are 11 classes of the wine quality (score between 0 and 10), which are determined from the 11 attributes. The attributes are the physicochemical feature of the wine, such as fixed acidity, residual sugar, pH, etc. There are 4898 instances in this data set.

This problem is interesting beause we can compare between randomized optimization algorithms and the back propagation method of the neural network, as well as compare among th randomized optimization algorithms themsevles. We will see that the strength of the back propagation is that it finds the direction of optimizing the weights directly from the errors of the outputs so it has the best performance. The weakness of the randomized optimization algorithms is that they have to find points randomly, so they are less efficient.

Here I used the randomized optimization algorithms to find the best weights which make the sum of squared errors mimimum: Randomized Hill-Climbing (RHC), Simulated Anealing (SA), Genetic Algorithm (GA).
Figure 1 (left) shows the relation between the sum of squared errors and the number of iterations of each algorithms. I used 10 hidden layer nodes in the network. From this figure we see that these three algorithms enventually converge at a mimimum sum of squared error of around 1200 after some large number of iterations. This is expected because all algorithms are supposed to find the (only one set of)

optimal weights. However, we noted that the SA needs much more iterations to converge than the other two algorithms. I think this is because SA initially started at very high temperature (10^11 in this case), so at first it is like completely random walk, which does not quite help to find the correct direction to the optima. After some iterations, the temperature deccreases and then it starts to climb the hills and find the optima. So it needs more iterations than other algorithms.

The computation time for different number of iterations is also show in Figure 1 (right). From this figure we can see that there is not much difference in the computatoin time of RHC and SA. But GA takes much longer times. This is because of GA does more work in each iteration: it needs to find the population, do the cross over, and mutations, while each iteration in the other two algorithm is relative simple: just jump to the other point randomly or with some probability.

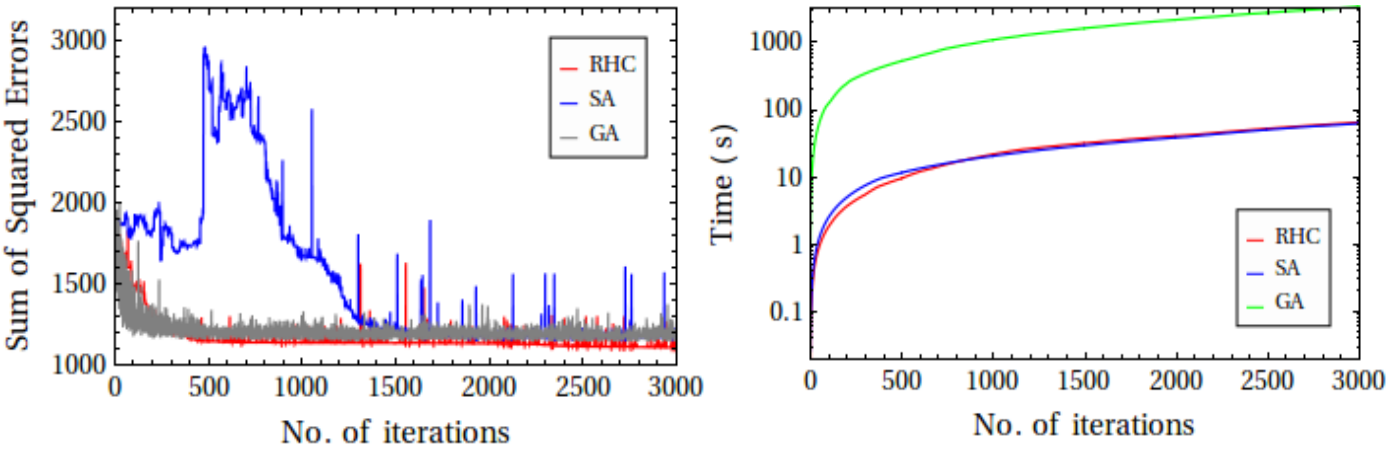

Figure 1: sum of squared Errors vs. number of iterations (left) and computation time as a function of iterations (right) of algorithms RHC, SA and GA for optimizing the weights of a neural network. 10 hidden layer nodes were used.

To better compare among the three algorithms, and with the back propagation from Assignment 1, I show in Figure 2 the accuracy and computation time of different algorithms after some number of iterations. Different algorithms are differentiated by different colors. The short solid lines represent accuracy or time of training data, while short dotted lines represent those of testing data. I got the the final result of the accuracy and time of back propagation from Assignemnt 1, which is independent of the number of iterations, but in order to make it easier to compare I also show them in the same figures.

Let's first look at the testing accuracy because it, rather than the training accuracy, is the one that really describes the accuracy of an algorithm. We see that for most number of iterations, GA has slightly higher accuracy than RHC and SA, though the three are actually very close. I think this slight better accuracy for GA is because it looks at the population sort of globably and do the cross over and select the best offsprings, so it can do a slight better work than the RHC and SA which mostly concentrate on some local points. However, when compared to the back propagation method of network, their accuracy is still lower than that of back propagation. This may be because back propagation method finds the weight in the opposite direction of the change of the output, so it knows the correct (or at least the good) direction to improve the weights, rather than trying different points randomly as in the randomized optimization algorithms. So it makes sense that the back propagation method still does a better work.

For the running time, the GA takes longest time, the reason of which was explained in Figure 1: GA does more complicated work in each iteration. The interesting thing is to compare the three randomized optimization algorithms with back propagation. We see the yellow solid line and the black dotted line at the very bottom of the Figure 2 (right), which means that the running time of back propagation is much less than the optimization algorithms. I think this is because the procedure of back propagation is much simpler than the optimization algorithms, it only needs to calculate the error of outputs and determine the direction of change of weights from that, which is straight forward and quite easy to calcualte, rather than looking for the next point or making pairs and mutate the points in the population in the optimization algorithms.

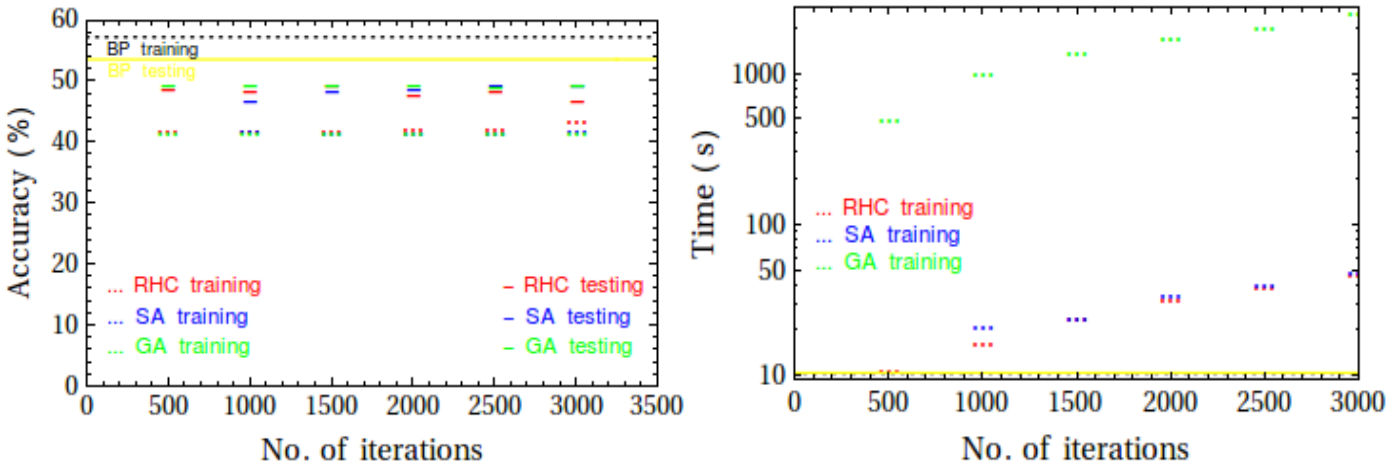

Figure 2: Accuracy (left) and computation time (right) of each algorith after some number of iterations, compared to the final results of back propagation obtained from Assignment 1. Different algorithms are differentiated by colors. Solid lines are for training data, and dotted lines are for testing data. 10 hidden layer nodes were used.

So my summary in this part is that for the neural network and this type of classification data set, it is more suitable to use back propagation method than to use the optimization algorithms. Because the former achieves higher accuracy in shorter time. But among the optimization algorithms, GA works slightly better but it takes much longer time, so there is a trade off between the accuracy and the time and which algorithm to use depends on the specific problem and the resources and constraints. we have.

2.2 Traveling Salesman Problem

The Traveling Saleman Problem [3] is to find the shortest possible route between several cities that visits each city exactly once and returns to the origin city. To better illustrate the result and consistent with other problems, the fitness function is defined as the inverse of reciprocal of the shortest distance. So we want to maximize it.

This problem is interesting. First because it is useful in our everyday life. For example, planning the best routes among several cities for delivery or postal serverices and transportation companies, or even planning the routes for a tourist who wishes to visit several cities with the mimimum time and cost. It is also interesting for theoretical studies, because it is a classical and well known optimization problem,

and has been proven to be an NP-hard. It can be used to compare behaviors of different algorithms in theoretical research and studies.

Here we study the behavior of the four randomized optimization algorithms under this problem. I would hightlight the advantages of the GA algorithm in this problem. The strength of the GA algorithm in this problem is that it can reach the optimal in fewer iterations than other algorithms. It has the weakness of relatively longer computation time but it is still acceptable.

Figure 3 shows the fitness function value and computation time of each algorithm as a function of number of iterations. We can see that GA can reach the optimal within least number of iterations. This is may be because GA looks at the population globly rather than at some local points like in SA and GA. It takes longer time than RHC and SA but still much shorter time than MIMIC. This makes GA the most practical algorithm to use for this problem.

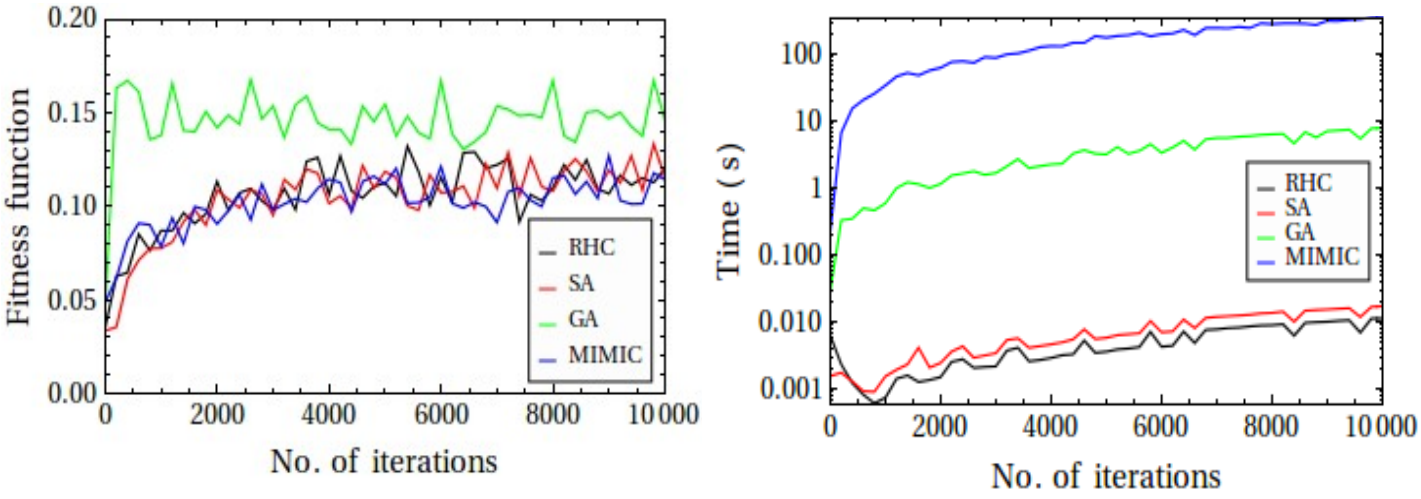

Figure 3: Fitness function value (left) and computation time (right) vs. number of iterations for the Traveling Salesman Problem.

We can make more investigations and modifications to improve the performance of GA. We do this in Figures 4-6.

Figure 4 shows the results of adjusting the number of population with other parameters fixed. We see that a larger population leads to smaller fitness function and takes longer computation time. So there is no need to use larger population in this problem. A population of 200 almost results in the maximum fitness fuction, yet with very short computation time.

In Figure 5, the number of mate in each iteration was modified while keeping other parameter the same. We can see that more mates results in larger fitness function but takes longer time. So 200 of mates can bring us almost optimal fitness function in a short time.

Finally, the number of mutate in each iteration was modified in Figure 6. It seems that the fitness function does not depend on the number of mutate monotonously but more mutate still needs longer time. But an interesting observation is that the numer of mutate which results in large fitness function also takes long time, and vice versa. So I would choose a mutate number of 25 and it still takes acceptable computation time.

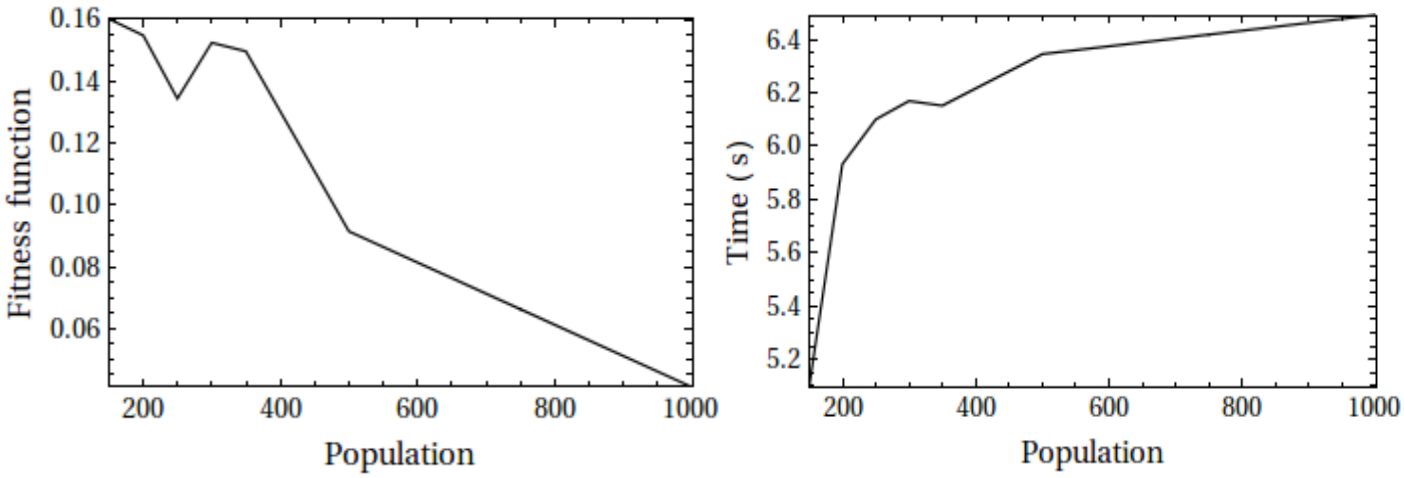

Figure 4: Fitness function value (left) and computation time (right) with different choices of population in the GA algorithm. The number of iteration was 10000. The number to mate and mutate each iteration was 150 and 25 respectively.

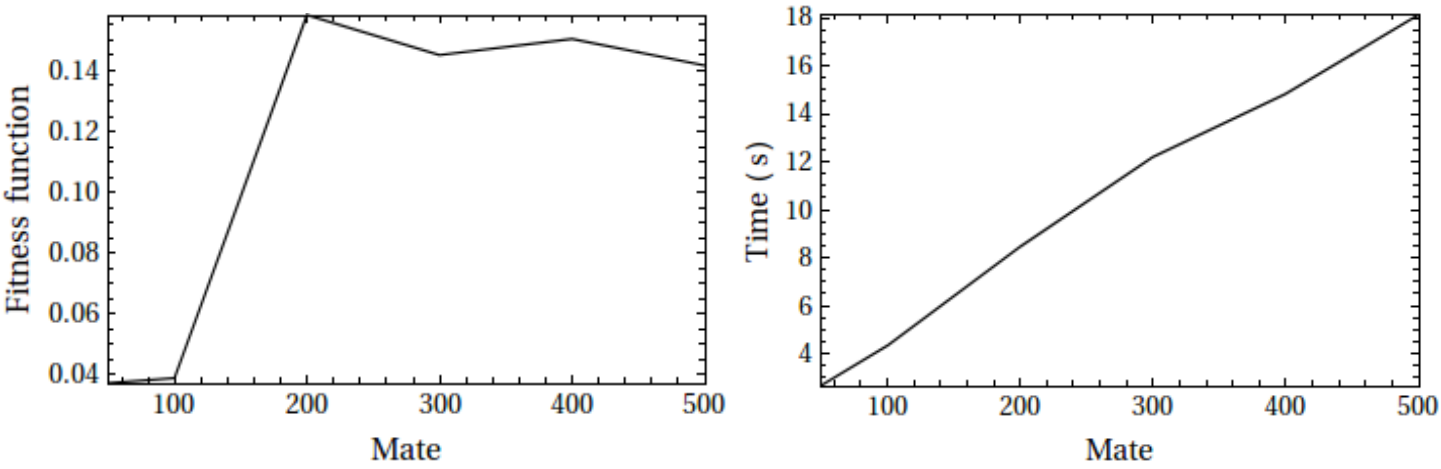

Figure 5: Fitness function value (left) and computation time (right) with different choices of number to mate each iteration in the GA algorithm. The number of iteration was 10000. The population was 500 and number to mutate each iteration was 25.

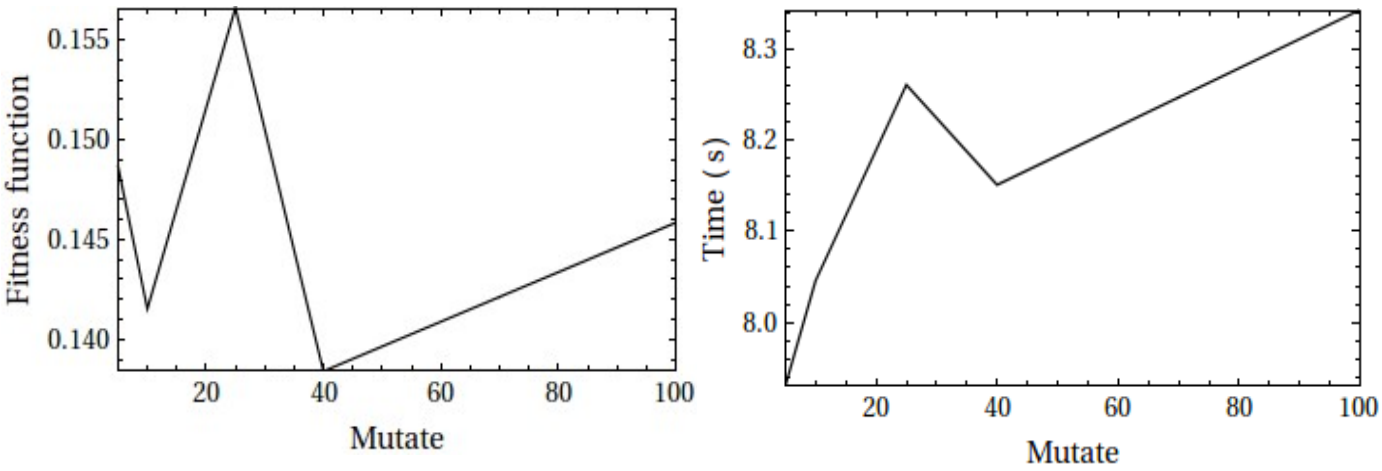

Figure 6: Fitness function value (left) and computation time (right) with different choices of number to mutate each iteration in the GA algorithm. The number of iteration was 10000. The population was 500 and number to mate each iteration was 200.

2.3 Four Peaks Problem

The Four Peaks Problem is described as below [4]:

The fitness function

$$f(x) = \text{Max}(o(x), z(x)) + \text{Reward},$$

where

$$z(x) = \text{number of contiguous 0's ending in position 100},$$
$$o(x) = \text{number of contiguous 1's starting in position 1},$$

$$\text{Reward} = \begin{cases} 100 & \text{if } o(x) > T \text{ and } z(x) > T \\ 0 & \text{else} \end{cases}$$

The four peaks problem is interesting because of theoretical studies. The fitness function has two global maxima and also two suboptimal local maxima. When T is large, this problem becomes more and more difficult because the basin of attaction for the local maxima become wider [5], making it easy to get stuck in those local maxima. Because of this property, it is a very good problem to use to compare the behaviors of different algorithms.

I would like to use this problem to highlight the SA algorithm. The strength of SA is that it takes shortest computation time due to its simplicity while reaching the maxima in the less iteration than RHC and GA. Its weakness is it does not converge so quickly as MIMIC. But considering both the number of iterations and the computation time, SA may be the most practical algorithm for this problem.

We compare how different algorithms reach the maxima of the fitness function with different number of iterations in Figure 7. The left figure is for small number of iterations, while the right figure is for

large number of iterations. We can see that MIMIC can reach the maxima within very small number of iterations, around 6500. This is consistent with what we learned in the lecture. But the other algorithms do not converge within 10k iterations, so I need to show their behavior with more iterations, which is in the right figure. From that we can see the SA reaches the maxima in slightly less iterations than RHC and GA.

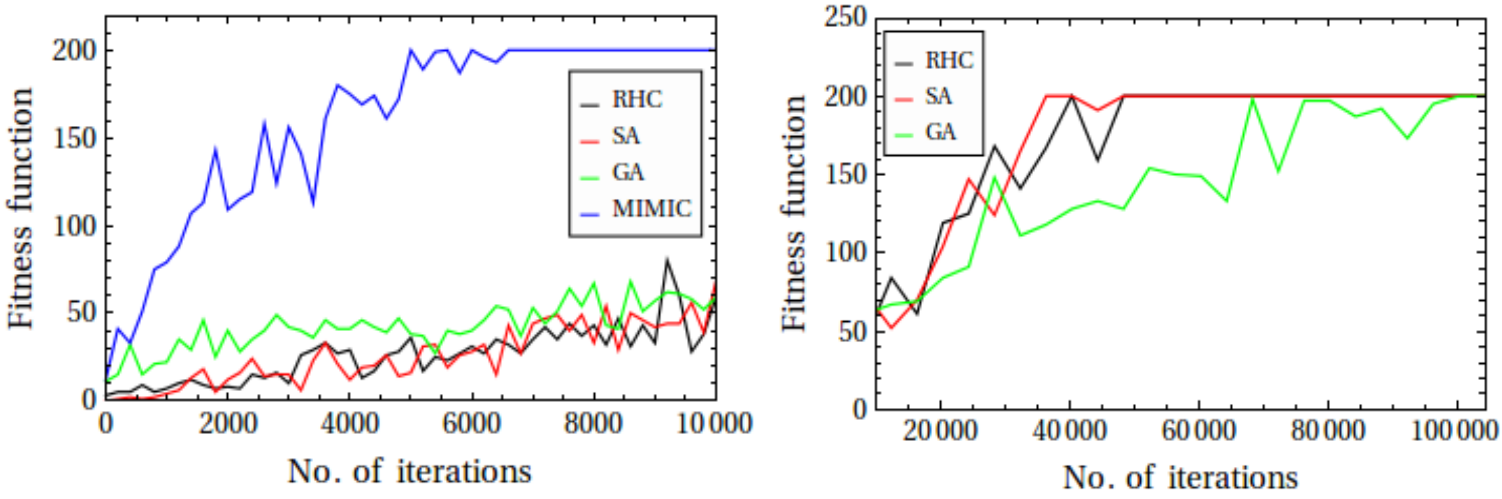

Figure 7: Fitness function values vs. number of iterations for the Four Peaks Problem. The left figure is for smaller number of iterations, while the right figure is for large number of iteations. Since MIMIC has already converged in the left figure, there is no need to show it in the right figure.

We then compare the computation time of the four algorithms in Figure 8. We see that MIMIC takes much longer time than other algorithms. SA and RHC takes least time. So consider both converging iterations and time, I think SA works best for this problem.

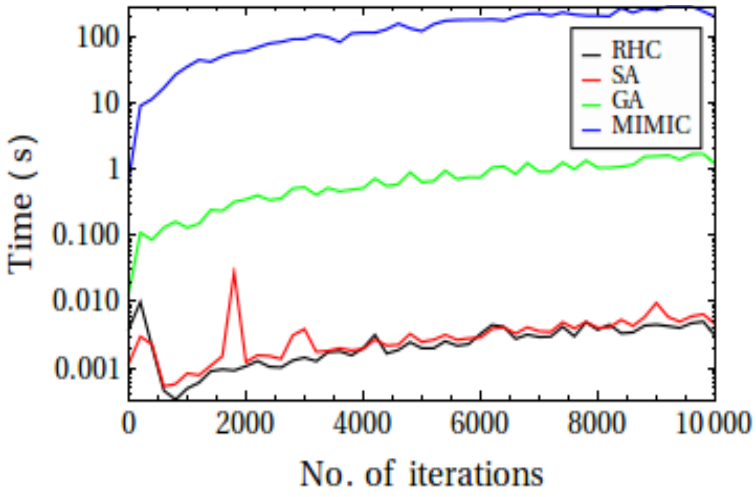

Figure 8: Computation time vs. number of iterations for the Four Peaks Problem.

Figures 9 and 10 show how we can improve the SA algorithm by making changes. We first change the starting temperature in figure 9. We see that the best starting temperature is around 1000. It takes the longest time but the time difference between different termperaure is actually small and thus neglectable.

We then change the cooling parameter in Figure 10. We see that larger cooling parameter results in larger fitness function. The computatoin time of different starting termperature also varies but their difference is still neglectable. So in practice we can just choose as larger cooling parameter as we can.

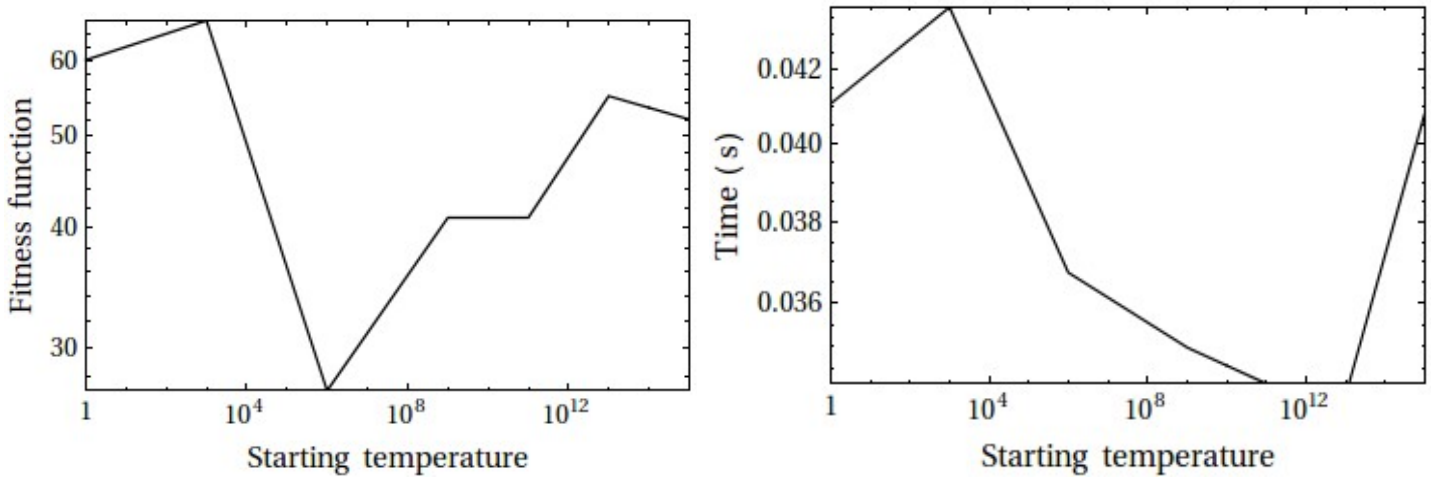

Figure 9: Fitness function value (left) and computation time (right) with different choices of starting temperature in the SA algorithm. The cooling parameter was fixed to 0.95. The number of iteration was 10000.

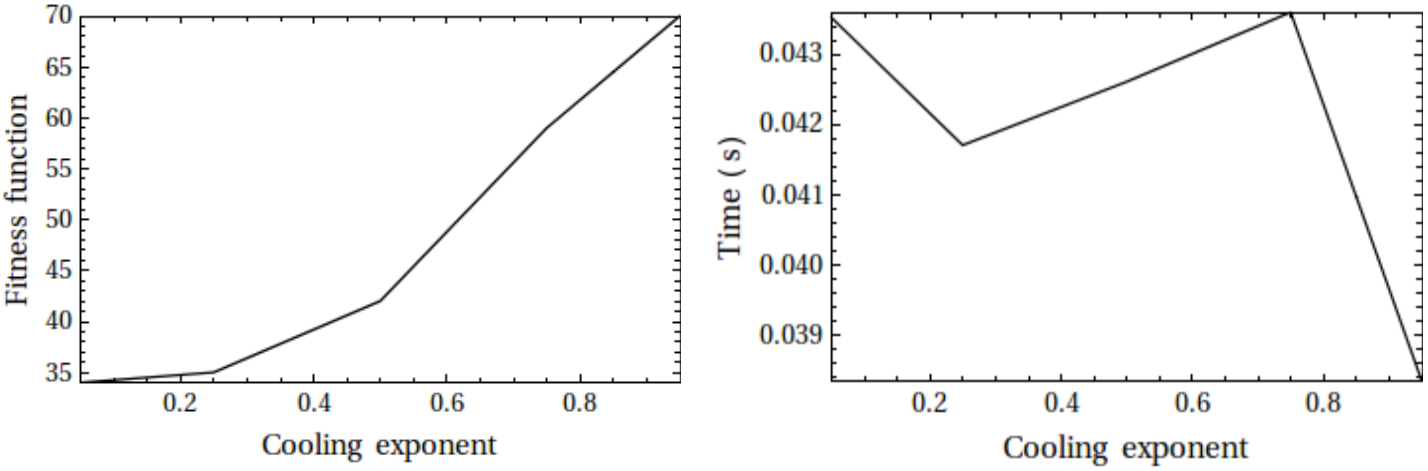

Figure 10: Fitness function value (left) and computation time (right) with different choices of cooling parameter in the SA algorithm. The starting temperature was fixed to 1000. The number of iteration was 10000.

2.4 Knapsack Problem

The Knapsack Problem [6] is described as this: Given several objects, each with a weight and a vlaue, determine the number of each objects to include in a collection so that the total value is largest while the total weight does not exceed a given limit. This problem is interesting because it is very popular and has many applications in finance, combinatorics, computer science and many other fields. Two examples are finding the optimal investments and portfolios in financial analysis and finding the least wasteful way to cut raw materials in industry.

I would like to use this problem to highlight the advantages of the MIMIC algorithm. The strength of the MIMIC algorithm is that it converges within smaller number of iterations, and the weakness is longer computation time.

Figure 11 shows the fitness function value and the computaton time with different numbers of iterations. We can see that MIMIC can reach the maxima within a very small number of iterations, and it does the best work among all the alogrithms. But its computing time is much longer than other algorithms. However, since 10000 iteration takes roughly 10 seconds, which is still acceptable. So in practice, I think MIMIC is the best algorithm to use for this problem.

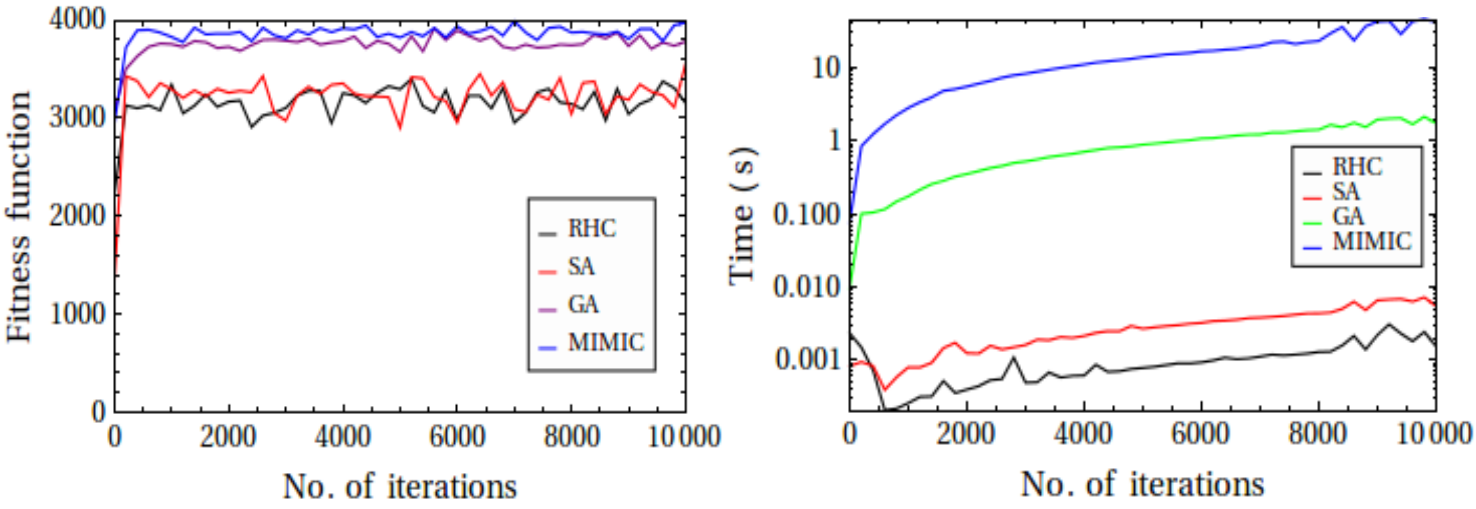

Figure 11: Fitness function value (left) and computation time (right) vs. number of iterations for the Knapsack Problem.

Figures 12 and 13 show what changes we can make to improve the MIMIC algorithm. In Figure 12, we changed the number of samples to generate. We can see that it is best to generate 300 samples. The more samples require longer time, but the time required by 300 samples is not bad. So in practice we like to use 300 samples.

Figure 13 shows changing the samples to keep. We see that in general, keep 10 samples gives the largest fitness function and takes almost the least amount of time. So we may use 10 samples in practice.

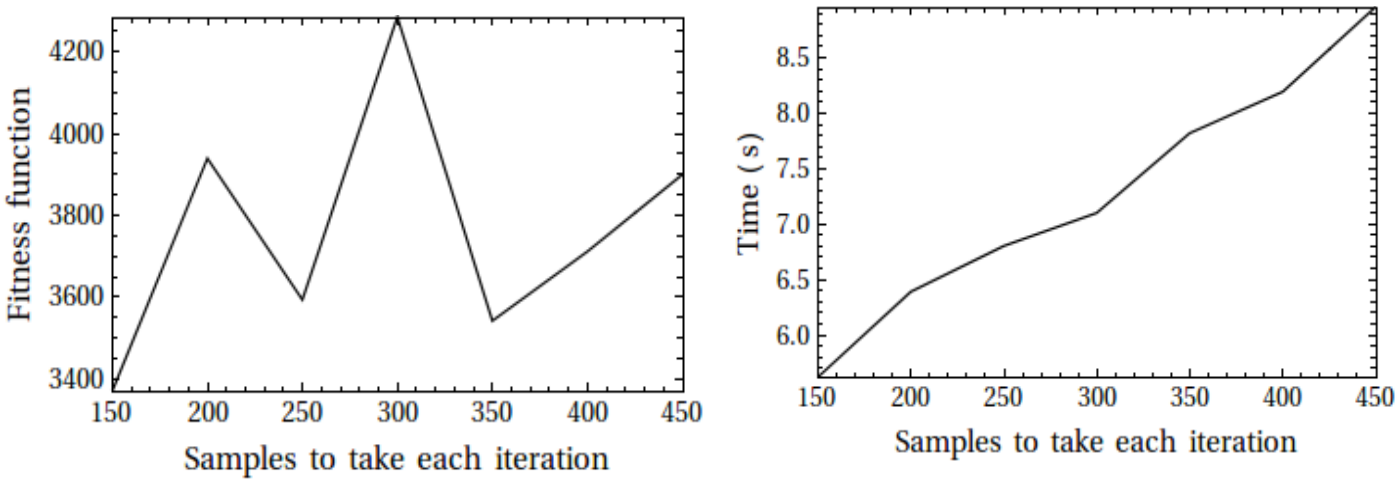

Figure 12: Fitness function value (left) and computation time (right) with different choices of number of smaples to take in each iteration in the MIMIC algorithm. The number of samples to keep was fixed to 100. The number of iteration was 10000.

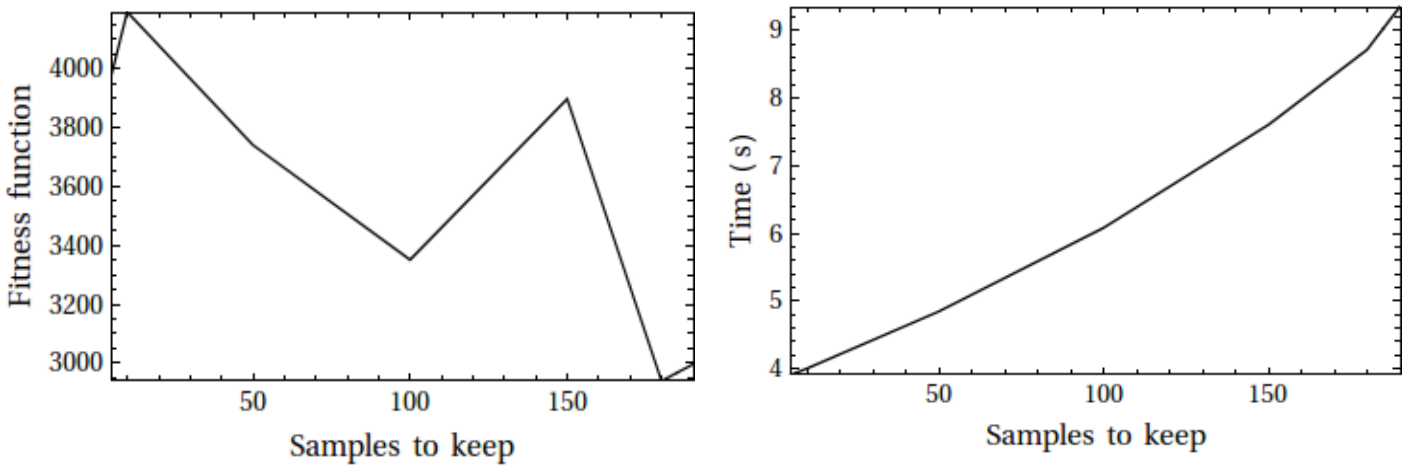

Figure 12: Fitness function value (left) and computation time (right) with different choices of number of smaples to keep in the MIMIC algorithm. The number of samples to generate was fixed to 200. The number of iteration was 10000.

3. Summary

In this assignment, we compared and studied the behaviors of four different randomized algorithms in four different optimization problems. For different problems, there is a different algorithm that works best, considering both the iterations and computation time. We also made changes to improve the behaviors of the algorithms.

References

[1] https://github.com/pushkar/ABAGAIL
[2] https://archive.ics.uci.edu/ml/datasets/Wine+Quality
[3] https://en.wikipedia.org/wiki/Travelling_salesman_problem
[4] S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. Technical report, Carnegie Mellon Univerisity, May 1995.
[5] Charles L. Isbell, Jr, Randomized Local Search as Successive Estimation of Probability Densities.
[6] https://en.wikipedia.org/wiki/Knapsack_problem