CS 7641 Assignment 2: Random Search Optimization

Zhengmao Liu--903208986

03/09/2017

1. Introduction

In this assignment, random search optimization algorithms were applied to neural network to find the optimized weights. Furthermore, three optimization problems were discussed to highlight the strength and weaknesses of different optimization algorithms.

Here is a brief introduction of the algorithms used in this assignment and their implementation.

1.1 Randomized Hill-Climbing (RHC)

RHC is a searching technique that looks for the global optimum by moving towards the next higher elevation neighbor until it reaches the global peak. By keeping track of the visited points, this algorithm could save much time and space from also randomly searching. By randomly starting at a different position, this algorithm is able to escape the local optima.

In this assignment, RHC was implemented in ABAGAIL java library. It was applied to the problems using Jython and the performance is compared with other algorithms in section 2.

1.2 Simulated Annealing (SA)

Simulated Annealing is inspired by metallurgy where metals are heated to high temperature and then slowly cooled to increase ductility. Therefore, in this algorithm, the temperature was set at a very high value and cooled down gradually with iteration. The optimization process is that the algorithm finds a next state S_{t+1} of the current state S_t, and determines whether goes to S_{t+1} according to a probability function. The iterative process is repeated until the system reaches a state that is optima or a given computation budget is exhausted.

SA algorithm was also implemented in ABAGAIL java library. It was applied to problems in both section 2 and section 3 using Jython.

1.3 Genetic Algorithms (GA)

Genetic algorithms are inspired by evolution methods, such as crossover and mutation. These algorithms initially start from a population of candidate solutions and continuously evolve to a better solution by eliminating non adequate sections of the population and retaining the parts of the population that exhibit the best traits. One advantage of GA is that it can skip the local minimum by mutation. One drawback is that GA does not scale well with complexity. If the population size is very large and the ratio of samples to keep and mutation is large, the algorithms would have to face an exponential increase in search space.

Genetic algorithms were implemented in ABAGAIL java library as well. Both section 2 and 3 have applied the algorithms and compared GA with other algorithms.

1.4 Mutual-Information-Maximizing Input Clustering (MIMIC)

MIMIC algorithm finds global optima by estimating probability densities. This algorithm keeps track of the structure of the optimization. Basically, it directly models the distribution based on the information from last iteration and refines the model according to probabilities in the population. MIMIC has the advantage of fast convergence. So less iteration could reach global maxima. The drawback is that the iteration would compute much longer than other algorithms.

MIMIC algorithm was implemented in ABAGAIL in java. The algorithm was applied to problems and compared with other optimization algorithms in section 3.

2. Optimization of Neural Network Weights

The Titanic data from Assignment 1 was used here to evaluate the three optimization algorithms, which are randomized hill climbing (RHC), simulated annealing (SA) and genetic algorithm (SA). These three algorithms were performed to find the optimal weights of a feed-forward neural network.

2.1. Introduction

The optimization of the neuron weights was implemented in Jython using ABGAIL.jar library. The Titanic data set was wrangled first in python, including converting categorical data to binary data, creating dummy features, cleaning missing data and so on. The iteration in the section was set at 5000 times. The results from Jython was saved to csv file and plotted by python.

2.1.1 Neural Network Architecture

The neural network architecture was from the assignment 1 for the titanic dataset, which has 8 neurons in the input layer, 4 neurons in the hidden layer, and 1 output layer. This structure was cross validated in Assignment 1 without overfitting problem. So the data was just split into 80/20 for training and testing, which are 834 instances for training and 209 instances for testing.

2.1.2 Back propagation and random search optimization

In back propagation, the target of optimization is to minimize sum of the squared error. The weights are updated by gradient descent. From the result in Assignment 1, the testing accuracy is 79% while the training accuracy is 82%.

In contrast, the target of the random search optimization here is to find the global maxima of a fitness function. The fitness function here is defined: $fintness = 1/error$. This function made it easy to update the weights and compare the performance of the optimization methods without much additional work. A comparison of the overall performance of test accuracy and computation time was discussed in following sub sections.

2.2. Randomized Hill Climbing (RHC)

RHC restarts randomly at a new location of the optimizing weight. The updating rule is similar to gradient descent. The difference is that RHC is updating one weight till the best value and then updating next. Gradient descent is simultaneously updating all the weights.

The errors and computation time of the classification (Figure 2.2.1, 2.2.2). The time was plotted in its logarithmic (base 2) value. Because of the randomness feature, the errors could not also converge to a smaller one as in back propagation gradient descent. But the advantage of this method is that the randomness could avoid the hill climbing process stuck in local minimum. The computation time was increasing linearly with the iteration.

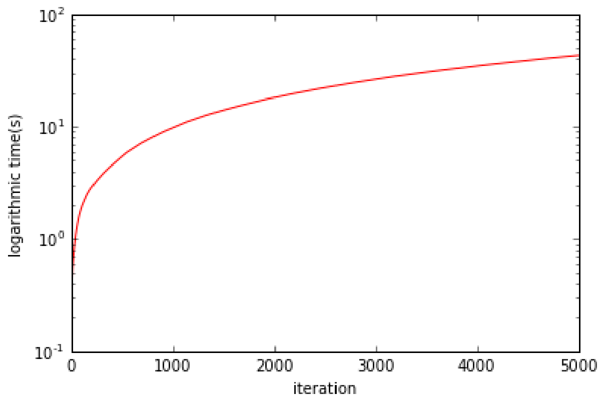

Figure 2.2.1 computation time of RHC

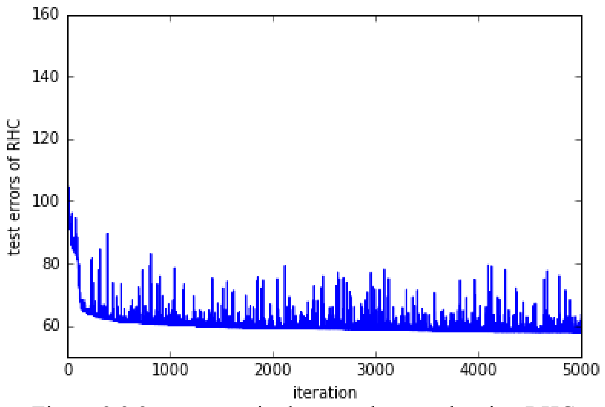

Figure 2.2.2 test errors in the neural network using RHC

Another note is that when the iteration is about 4000 times, this optimization seemed to find the weights close to the optimal value.

After applying the RHC optimized weights to the neural network, the training accuracy is 81%; while the testing accuracy is 78%. This result is very close to the result from assignment 1 when using back propagation.

2.3. Simulated Annealing (SA)

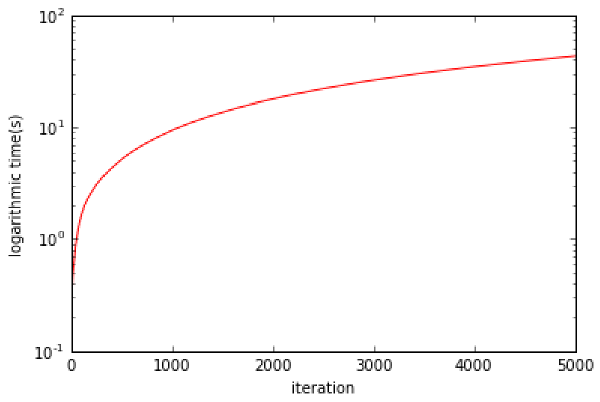

Figure 2.3.1 computation time of SA

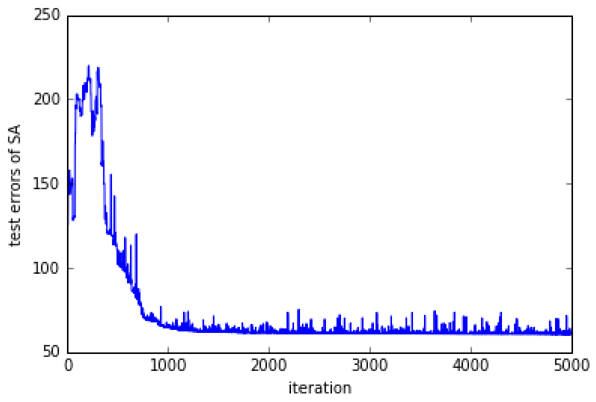

Figure 2.3.2 test errors in the neural network using SA

Different from RHC, SA did a global random search from the beginning. At the small iteration times, the tempuerature T was very big, so the performance was more like a random walk according to the probability criteria in SA. This can be found from the Figure 2.3.2 when the iteration is less than 500 times. With the increasing of iteration, the temperature T went smaller, the process acted like hill climbing but still able to skip the local optimal.

Because of the random search, SA was able to reach the optimization results with much less iteration. After about 2000 to 3000 iteration, the algorithm was able to reach the maxima. The computation cost (Figure 2.3.1) was in the same magnitude as the RHC.

Different cooling exponent was also studied. Since the cooling exponent decided how fast the temperature would calm down, I mainly tested the values at 0.85, 0.9 and 0.95. The testing accuracy and computation time cost are very close. When cooling exponent equaled to 0.9, it has slightly better result in accuracy.

Applying the optimized weighted in the neural networks, the testing accuracy is 78% while the training accuracy is 80%. This result is also very close to the outcome from the RHC and Assignment 1 using back propagation.

2.4. Genetic Algorithm (GA)

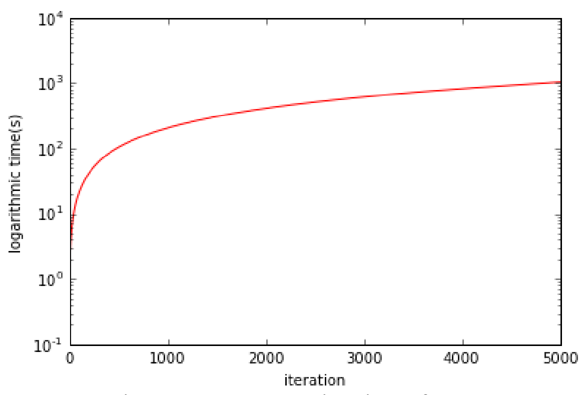

Figure 2.4.1 computation time of GA

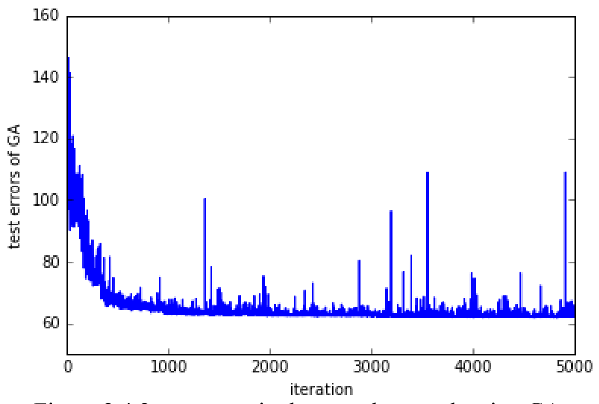

Figure 2.4.2 test errors in the neural network using GA

GA algorithms are very different from RHC and SA. It keeps a set of hypothesis and then evaluates each hypothesis using the fitness function. A new set of hypothesis would be generated by operating on the previous hypotheses, such as crossover and mutatation. This characteristic of GA makes it take much longer time than both RHC and SA to do iteration. The computation time is one magnitude more than that in RHC and SA (figure 2.4.1). Since GA always chose good hypotheses from a portfolio, it has much less randomness, which is offered by the ratio of mutation, than RHC or SA (Figure 2.4.2).

Parameters of GA that are to be tuned include the population of hypothesis size, the size of samples to keep in the new generation, and the mutation rate (in the following list).

Population size	[200, 400]
Ratio of Samples to keep	[0.4, 0.6, 0.8]
Raito of mutation	[0.001, 0.01, 0.05]

The difference from tuning of the above parameters in test errors was very small. The computation time is proportional to the population size. However, the accuracy of applying the weights to the test set was quite big. The following table shows the comparison of the accuracy.

Population size	Ratio of samples	Ratio of mutation	Training accuracy	Testing accuracy
400	0.4	0.001	79%	78%
400	0.6	0.01	80%	78%
400	0.8	0.01	80%	77%
200	0.4	0.01	78%	74%
200	0.6	0.05	78%	74%

The final parameters were chosen as population at 400, ratio of samples to keep was 0.5 and the mutation ratio was 1%. So the training accuracy is 80% and testing accuracy is 78%. This result is similar to those from back propagation, RHC and SA.

2.5 Comparative Analysis

The best algorithm for this problem set should be the one that has nearly best accuracy when applying to the neural nets and also has fast computation time.

Comparing the above results with those from back propagation, one can conclude that for this Titanic dataset, back propagation, RHC, SA and GA have similar performance in optimizing accuracy when using a simple one-hidden-layer neural network. In terms of their computation time, GA is not a good choice because it took too much time. SA was able to reach the global maxima with less iteration. So SA or back-propagation could be used for this problem.

3. Optimization Problems

In this section, three optimization problems, four peaks, continuous peaks, and Knapsack, were considered to evaluate the strength and weakness of different optimization algorithms. In each problem, the four algorithms were analyzed and compared by their fitness function values and computation time first. After that, the best algorithm for each problem was tuned to achieve the best optimization. The analyzing part was programmed by Jython using ABAGAIL.jar library. Each program was run for 5 times and the results were averaged to conduct the analysis.

3.1. Four Peaks Problem

3.1.1. Description

This four peaks problem was defind from (Baluja and Caruana, 1995). This is a bit string optimization problem. Given an N-dimensional input vector $\vec{X}$, the four peaks evaluation function is defined as:

$$\boldsymbol{f(\vec{X},T) = \max[tail(0,\vec{X}), head(1,\vec{X})] + R(\vec{X},T)}$$

Where:

$$tail(0,\vec{X}) = number\ of\ trailing\ 0's\ \ in\ \vec{X}$$

$$head(0,\vec{X}) = number\ of\ leading\ 1's\ in\ \ \vec{X}$$

$$R(\vec{X},T) = \begin{cases} N & \text{if } tail(0,\vec{X}) > T \text{ and } head(1,\vec{X}) > T \\ 0 & \text{otherwise} \end{cases}$$

Two global maxima are there for this function. They are achieved either when there are T+1 leading 1's followed by all 0's or when there are T+1 trailing 0's preceded by all 1's. There are also two suboptimal local maxima that occur with a string of all 1's or all 0's. When T becomes larger, this problem is more difficult because the basin of attraction for the inferior local maxima become larger. T was set at N/5 here. N was set at 400 to make the problem more complex.

3.1.2. Analysis

A comparison of fitness function value and computation time based on iterations was performed first (Figure 3.1.1, 3.1.2). Genetic algorithm has the best fitness with moderate computation time. Genetic algorithm is highlighted here. GA can reach the global optima with its crossover and mutation functions. In comparison, the structure of the optimization landscape is not critical in this optimization. Though converging fast, MIMIC is not able to find the global optimal. For random hill climbing and simulated annealing, it looks like they were stuck in the local minimum because of the large gap of local maxima as described above.

Additional studies were performed to analyze the factors of population size, crossover size and mutation size on the performance for this problem (Figure 3.1.3 to Figure 3.1.5).

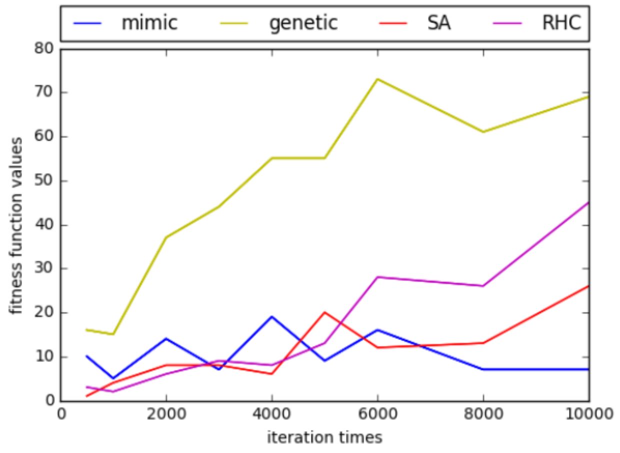

Figure 3.1.1 Fitness value comparison

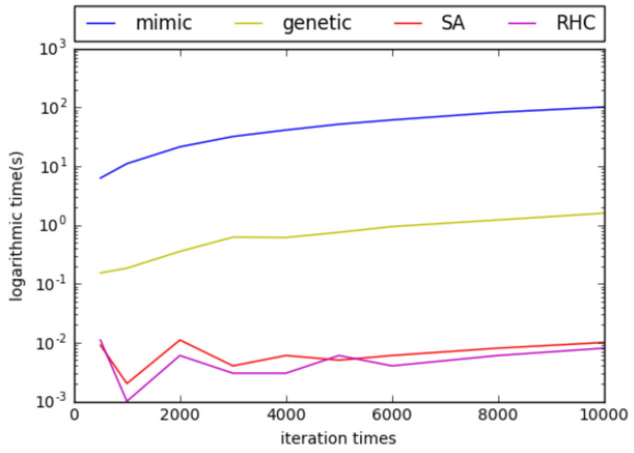

Figure 3.1.2 computation time comparison

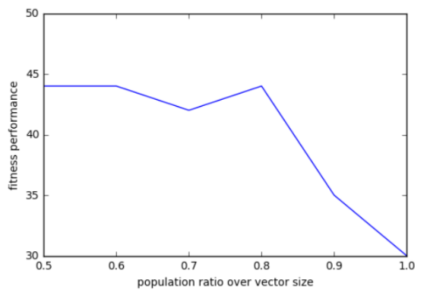

Figure 3.1.3 fitness on population size

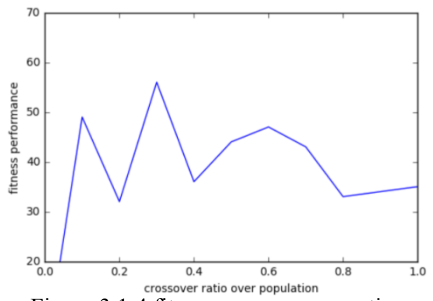

Figure 3.1.4 fitness on crossover ratio

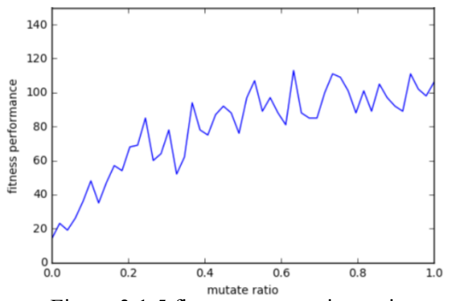

Figure 3.1.5 fitness on mutation ratio

Given fixed ratios of crossover and mutation, GA performed best when the population hypothesis was set about 0.5 to 0.6 (figure 3.1.3). If the population was too big, GA would be unable to select the hypothesis with high fitness probability. For the crossover ratio, the fitness could reach its maximum when it is about 0.3 to 0.4. This can guarantee that the hypotheses with high probability from the last step can still remain in the population set while new set of hypotheses is generated. For the mutation rate, it is very interesting to see that when the mutation ratio is as high as 0.5 to 0.5, the fitness performance could improve a lot. This is mainly because that the mutation operation is the key in this problem to escape the large gap of local maxima.

3.2. Continuous Peak Problem

3.2.1. Description

The continuous-peaks problem is based on the four peak problem (Baluja and Davies, 1997). Rather than forcing 0's and 1's to be at opposite ends of the solution string, they are allowed to form anywhere in the string. A reward is given the there are greater than T continuous bits set to 0 and greater than T continuous bits set to 1. Therefore, instead of the sum of two global maxima and two suboptimal local maxima, the target of this problem is to find the global peak among all of the local peaks (Figure 3.2.1).

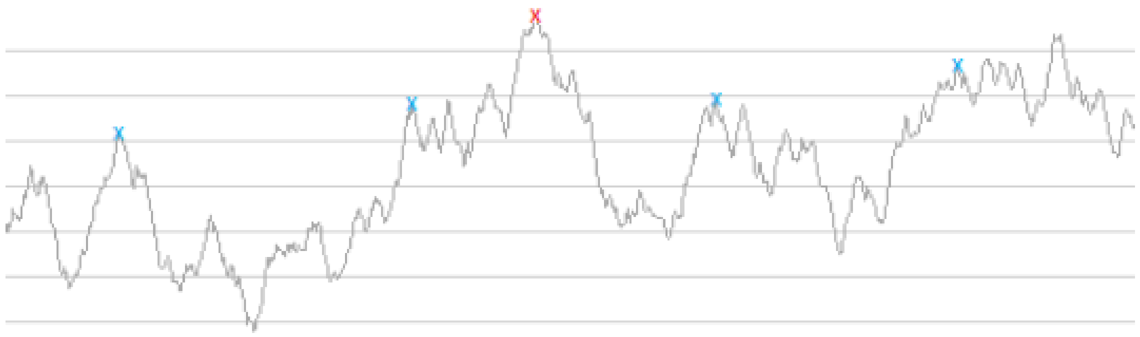

Figure 3.2.1 Continuous peaks problem

The interesting part of this problem is that it has many local maxima, so the normal hill climbing would highly probably fall in any of them. Additionally, unlike the four-peak problem, this does not specify the location to be tail or head, so the genetic information from "parent" genes would not be very useful here. In fact, the genetic algorithm might not be converging at all. The simulated annealing algorithm is supposed to work very well in this problem.

3.2.2. Analysis

Simulated annealing outperformed RHC and GA in fitness functions as expected previously (Figure 3.2.2). Within a small number of iterations, SA did not have similar performance as MIMIC, but after 2500 iteration times, SA was very good in fitness. MIMIC algorithm could perform very well is mainly because the searched space was stored by the MIMIC algorithm, these visited space made it easier and much faster to converge to the global maxima.

Compared with MIMIC, the computation time of SA is two magnitudes less (Figure 3.2.3) in this problem. Considering both of the fitness performance and computation time, SA is the best optimization approach here.

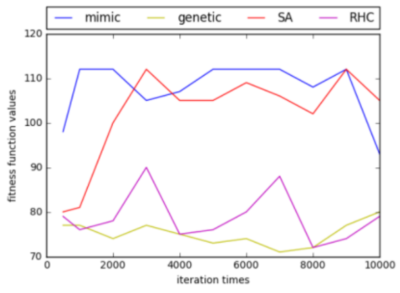

Figure 3.2.2 Fitness value comparison

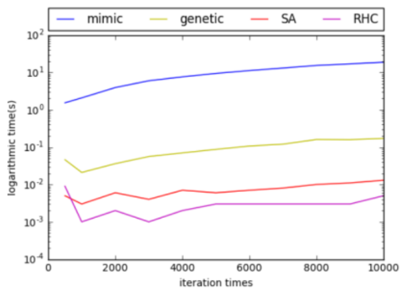

Figure 3.2.3 Fitness time comparison

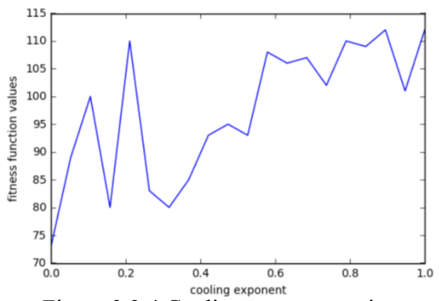

Figure 3.2.4 Cooling exponent ratio

The studies of cooling exponent parameter were performed (Figure 3.2.4). The iteration time here was set at 8000 from previous result. When the cooling exponent was 0.85 to 0.9, SA optimization could perform best here. Because T is updated by $T \cdot Rate_{cooling}$, if T was cooled down too fast (small cooling exponent), SA would be very similar to hill climbing, which would be stuck in local optimal. This is why the fitness value has a big variation at small cooling exponent (Figure 3.2.4). If the cooling rate is very big, SA would always be like a random walk. It might not be converging at all.

3.3. Knapsack Problem

3.3.1 Description

The knapsack problem is a problem of combinational optimization: given a set of items, each with a value and a weight, the objective is to find the number of each item to be included in a collection so that the total value of the collection is maximized constrained by a limited total weight. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items.

The maximum possible value of the collection is set at 4000. There are 40 items and each has 5 copies. Maximum weight and volume per item are set at 50 units.

This knapsack is a typical NP-hard programming problem; the optimization state of iteration step K is totally based on the iteration state (K-1). The optimization at state K is a sub-problem of optimization at state (K-1). So we need to keep track of the structure of the optimization landscape here. MIMIC outperforms other algorithms here with much less iteration to reach optimal state (Figure 3.3.1, 3.3.2)

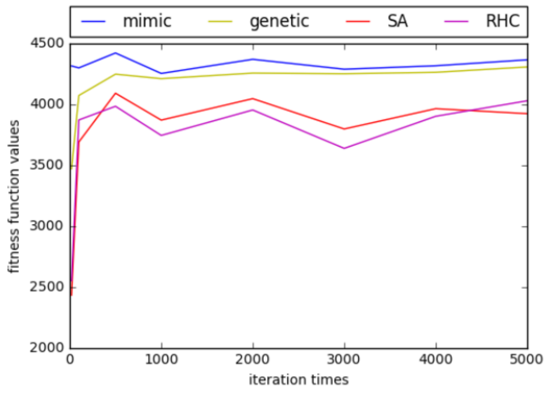

Figure 3.3.1 Fitness value comparison

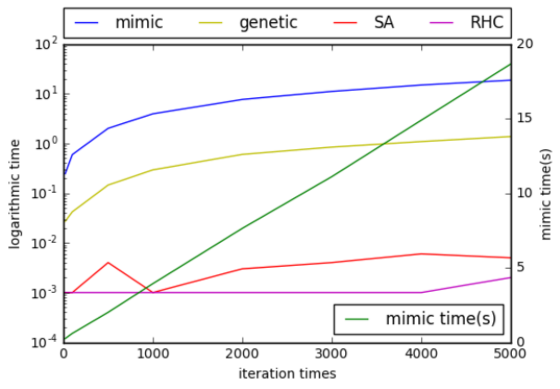

Figure 3.3.2 computation time comparison

3.3.2. Analysis

From the above diagrams and results, we can see that the convergence of MIMIC regarding the iteration is very fast (Figure 3.3.1). When the optimization only iterated less than 50 times, MIMIC algorithm has already reach its best fit. The performance of other algorithms might also improve with the iterations, but still poorer than MIMIC.

It is also interesting that the genetic algorithm also performs comparably to MIMIC. This is mainly because that the crossover operation of genetic algorithm would have similar behavior as sampling from a distribution, such that the genetic algorithm can match the underlying structure.

MIMIC is the best algorithm here is that it can compensate the computation time cost by doing much less and reaching the same fitness (Fig 3.3.2). For example, GA had to make its best optimization after more than 1000 iteration. The computation cost of this 1000 iteration was about 0.3 seconds. But MIMIC could reach a better fitness by doing 50 iteration times, which also took about 0.3 seconds. So MIMIC still had better overall performance in time and fitness.

Additional studies were focused on the sample size and the population size to see the influence of population size and sample size to the overall performance. Both fitness function values and computation time were compared at different sizes. The iteration number was fixed at 100. The computation time is exponentially increasing with the population size. But when the population size is greater than 120, the increase of fitness values starts to be flatter. If there is no limit in computation time, the population size could be set to 200 to achieve the best fitness performance. Otherwise, 120 could be a good choice.

The sample ratio in the population can mostly optimize the performance when it was 0.5 to 0.7. If the ratio was too big, there would be fewer new hypotheses with high probabilities to be selected. If the ratio was too small, some hypotheses with good fitness would have to be dropped out. Then MIMIC would be unable to keep track of enough information of the path and the previous states. So the optimization would be not that fast to converge.

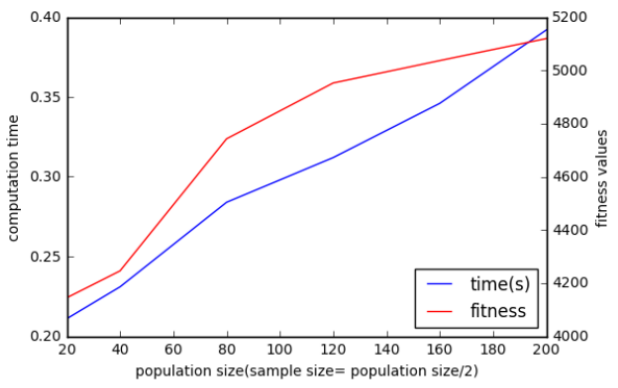

Figure 3.3.3 population size with the fitness function

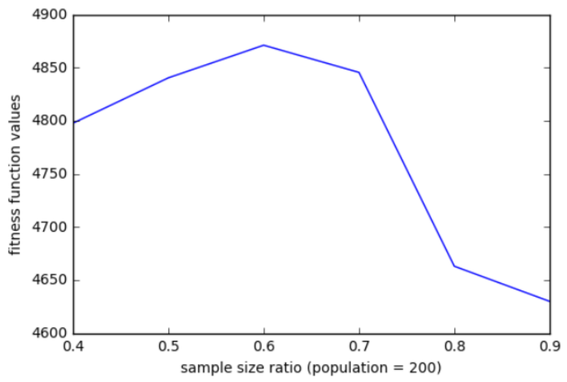

Figure 3.3.4 sample ratio

4. Conclusion

In this assignment, four random search optimization algorithms, RHC, SA, GA and MIMIC, were studied.

RHC, SA and GA were compared with back propagation in optimizing neural network weights. The parameters for these algorithms, such as cool exponent in SA, population size, sample to keep size and mutation size in GA were also tuned and studied. For a simple network structure used in this assignment, RHC, SA, and GA all have as good performance in reducing errors as back propagation. However, GA took too much computation time compared with others. SA and back propagation would best fit here because they could reach the minimum error performance within shortest time.

SA, GA, and MIMIC were applied to different optimization problems in section 3. GA is the best algorithm for the four peak problem because GA has the advantage over bit spring data and optimization strategy in GA can make the best fitness for four peaks. SA is the best for the continuous peak problem in terms of both the fitness function and computation time. MIMIC is close to SA in terms of fitness function, but very slow in computation. MIMIC is the best for knapsack problem because MIMIC keeps track of the information from iteration, which is very important to find the global maxima in NP hard problems like knapsack. The drawback of MIMIC, slow computation, can also be compensated by doing less iteration.

5. References

1. Bajuja, S. and Caruana, R. (1995). Removing the genetics from the standard genetic algorithm. Technical Report, Carnegie Mellon University
2. De Bonet, JS., Isbell C, and Viola P (1997). MIMIC: Finding Optima by Estimating Probability Densities. Massachusetts Institute of Technology
3. Bajula, S. and Davies, S., (1997). Using Optimal Dependency-Trees for Combinational Optimization: Learning the structure of the Search Space