Random Optimization problem

Abstract

This assignment report will explore 4 different kinds of popular random optimization algorithms – for both continuous variables problem and discrete variables. The first part of the report will optimize the weights of the neural network established in assignment 1 and second part of report shows three discrete optimization problem: continuous peaks, flip-flop problem and traveling sales problem.

Introduction

- Randomized hill climbing, refer as RHC in the rest of the report, is an algorithm trying to find a better solution by making incremental change. This algorithm will randomly select a starting point to avoid stuck at local optimal.
- Simulated annealing, refer as SA in the rest of the report, is an algorithm get inspiration from the physical processing of heating metal and then cooling down to reduce defect. It tries to find the solution by moving the step according to a probability function $P(x_t, x_{t+1}, T)$, in which $P(x_t, x_{t+1}, T)$ is defined as:
  If $f(xt+1) >= f(t)$, then $P(x_t, x_{t+1}, T) = 1$

  Otherwise, $P(x_t, x_{t+1}, T) = e^{\frac{f(xt+1)-f(xt)}{T}}$

  In the equation, f(x) is the fitness function of the given problem. T is a given temperature and will decrease with iterations
- Genetic algorithm, refer as GA in the rest of the report, is an algorithm that iterates a population of solutions and output the solution from the population. In each iteration, a given size of population is selected from hypothesis space according to the fitness function. Select the most fitness individuals and replace the least fit individuals via crossover and mutation.
- MIMIC, which refers to Mutual-Information-Maximizing Input Clustering, is an optimization algorithm that mainly contains two steps in each iteration: 1. Output a population of hypothesis that is most likely to contain the optimal solution according to a probability estimator 2. Estimate the input hypothesis structure with the probability estimator based on the population in the previous step.
- Java package Abagail is used to implement all optimization algorithms.

1. Neutral Network weights optimization

Dataset

In this report, we continue to use the Austin Animal Shelter data to evaluate different kinds of algorithms. The original task is a multi-classification problem. To make the comparison of the optimization algorithm more efficient, the problem is changed to a binary task – when the pet is 'adopted' or 'returned to owner' we label is as 1 (means the animal has a good result); when the pet dies or is transferred to other places (means that animal are not adopted in the end). This transformation also fit our need from very beginning – to predicting whether the animal is adopted

so that we can allocate resources accordingly.

To ensure the comparison is efficient, we redo the hyper-parameter tuning for assignment one. Different parameters are chosen in init-learning rate, max iteration, network structure – how many hidden layers is required, how many units is required in each hidden layer. The optimal structure is a neutral network with two hidden layers of 100 units. With back propagation, we can use the ADAM optimizer (with momentum). This usually makes the training much quicker (less than 200 iterations). We can clearly see that after 170 iterations with 0.87 F1 score in the testing data set, the total test data cost doesn't decrease much any more. We are going to compare the results with RHC, SA, GA and back propagation in later sections.

1.1 Overall performance at first 500 iterations

We want to compare the performance of difference algorithms. In this section, we will include the following algorithms: RHC, SA and GA on the training dataset in the first 500 iterations. The result is compared to back-propagation. The comparison is shown in figure 1.1. Back-propagation gets the best performance. It goes well beyond 0.83 in f1 score while RHC and SA stay low in f1 score at 0.5 to 0.6 (we will analyze the performance of RHC/GA within more iterations in later sections). GA performs a little bit lower than backprop but stay relatively stable after 200 iterations. Please note that all algorithms try to optimize the mean square error (cost function), but figure 1.1 shows the F1 score.

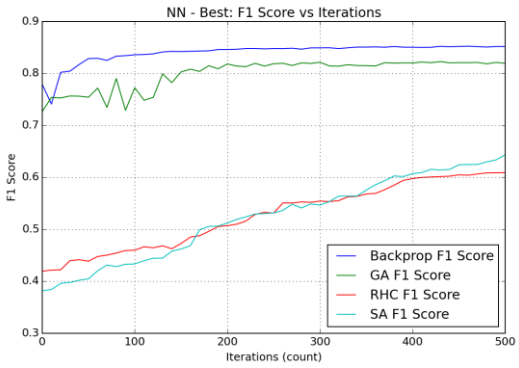

Figure 1.1 F1 score across first 500 iteration for backprop, GA, RHC and SA

1.2 GA

The final f1 score got from GA is around 0.82 on the testing data. The training/val/testing result is very similar to each other. The performance (f1 scores) doesn't change much after 200 iterations. GA is taking relatively long to run – 15 mins for every 10 iterations and takes about 12.5 hours for 500 iterations. There are three parameters to tune for GA – population size, numbers to mate and number to mutate.

Population size determine the population to check in each iteration. If population size = 1, then GA is working like the RHC.

Number to mate determines how many pairs we want to choose for crossover. The offspring from the crossover will bring to the next iteration.
Number of mutations determines how many we want to mutate in the rest of population before putting into new populations. Increase any of the three parameters will increase the running time for GA but will improve the performance.
To find the best weights for the network, the following two charts show a comparison on the mate parameter (mate pairs of 10 vs 20). With mate = 10, after 500 iterations, the f1 score stuck at 0.77 and cannot improve any more, while with mate = 20, the f1 score easily get around 0.82 after 200 iterations.

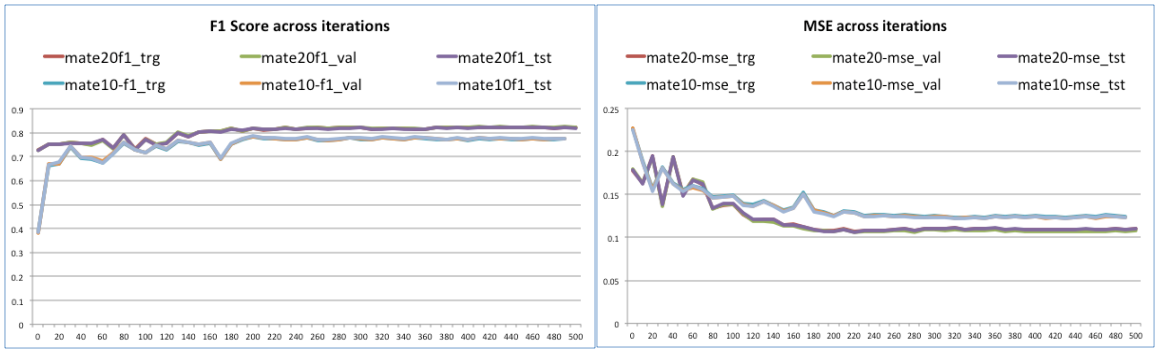

Figure 1.2 Analysis of GA, F1 score (left), MSE (right)

1.3 RHC

In the overall section, we saw the f1 score continue to increase after 500 iterations, therefore we increase to 5000 iterations to check how well RHC performs. Figure 1.3 shows that the f1 score increases to 0.82 after 5000 iterations. The improvement speed slow down after 2700 iterations and but will continue to run even after 5000 iterations.

Overall, every 10 iterations take about 40s seconds (much faster than GA) for RHC. This is because GA needs to evaluate the fitness function (cost function) for a population, while RHC only needs to evaluate for the next potential better solution. To get a better result, we can simply improve the number iterations. However, for really large network, this may require significant computation resources.

Figure 1.3 Analysis of RHC, running time (left top), F1 score (right top), MSE (left bottom), accuracy (right bottom)

1.4 SA

Just like RHC, SA algorithm's f1 score continue to increase (mse continue to decrease) after 500 iterations, therefore, we increase the in interaction to 5000 iteration. The SA algorithm will get around 0.83 f1 score at 5000 iterations. The most important factor for SA is the temperature, T. If the T is setting to be too low, then SA just works as Hill climbing. If T is setting to be too high, then SA will do random walk. In the following charts, we compare the performance of SA for different kind of T setting (we show f1 core for training/validation/test, but they overlap with each other because of small differences).

When T is higher, the running time of each iteration takes longer. This is due to the fact we need to calculate the probability ($\exp((f_{t+1}- f_t)/T)$) more frequently at higher T. In our animal shelter example, 5000 iterations takes 2500s when T = 0.9 but only 1500s when T = .15. However, this time difference generally is relatively small, and performance rather than time complexity should be the top concern for SA.

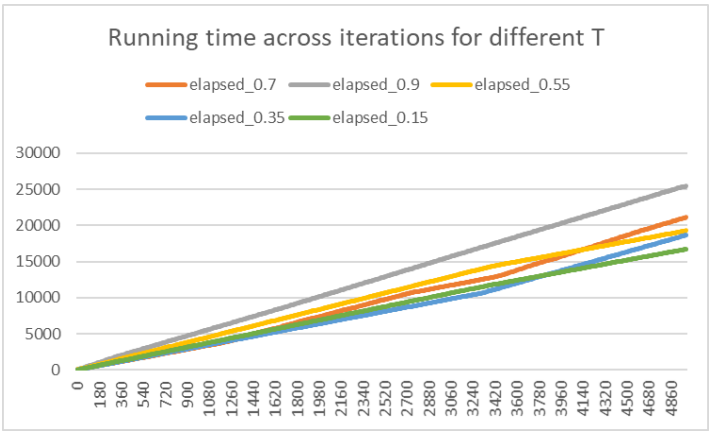

Figure 1.4.1 Running Time of SA for different temperature setting

It is clear to see that when T = 0.95, the algorithm is more likely to wander around and have back and forth than other temperature setting (dropped in the first few hundred iterations). When T = 0.15, the algorithm climbs slowly to a higher f1 score but underperform than other temperature. T= 0.55 seems to be the best parameter among all temperature setting, converging quickly around 1800 iterations. Choosing an appropriate temperature is the key to get SA GA works better.

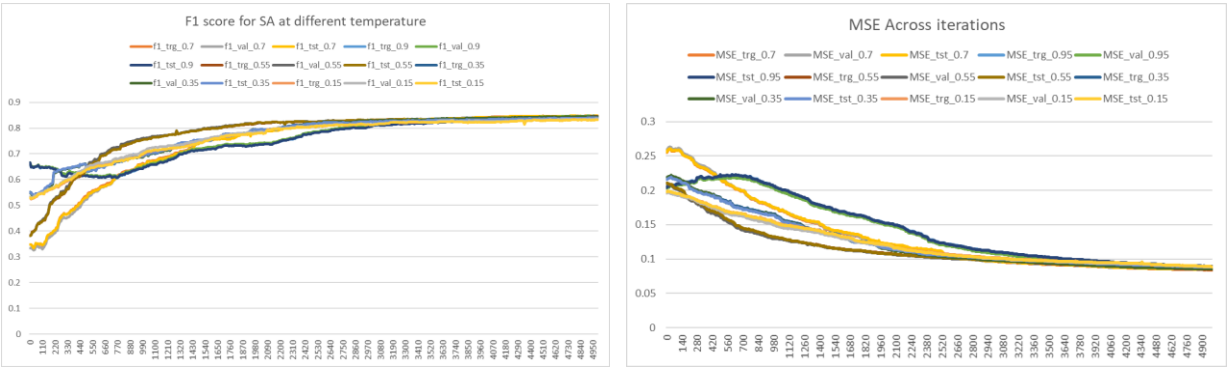

Figure 1.4.2 Analysis of SA F1 score (left), MSE (right)

1.5 Backprop

We found that the MSE (the setting cost to optimize) continue to decrease after 800 iterations, the test f1 score doesn't improve after first few hundred iterations. Figure 1.5 shows that the training f1 score continue to reach around 0.89 or higher, which indicate overfitting on the training data set.

The best way to optimize backprop is to use Adam. It uses exponent average/momentum and adjust the learning rate automatically. Running time can also be greatly reduced because of quicker converging (less iterations is needed)

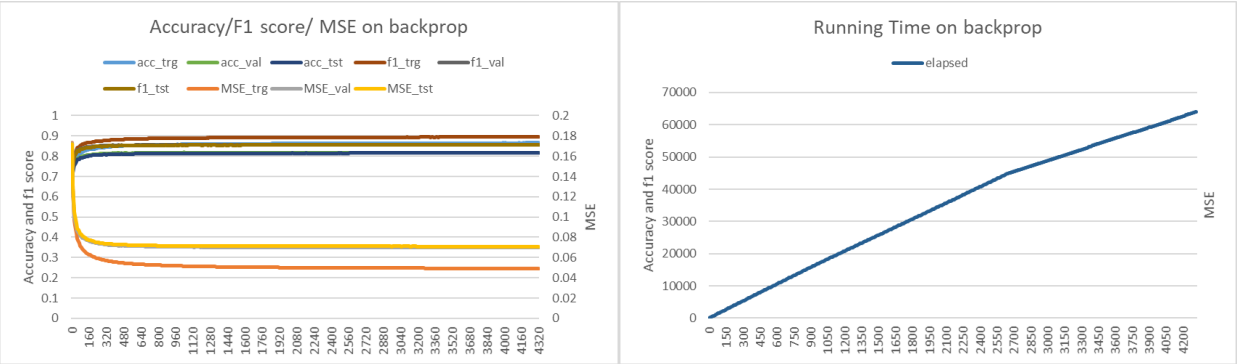

Figure 1.5 Analysis of back propagation, F1 score/MSE /accuracy (left), running time (right)

1.6 Conclusion on NN weights optimization

In conclusion, all the 4 randomized optimization algorithms reach above 0.80 at f1 score given enough iterations (the performance on the testing dataset is showing as below). However, none of them can guarantee reaching the global minimum instead of local minimum. Backpropagation are the most efficient (the converging is the fastest), thus the most commonly used algorithm in the industry for neutral network. This is not surprising, since backpropagation provides the direction that the weights can be optimized to and the use of ADAM will enable the weight movement even more efficient. All other optimization algorithms (GA, RHC, SA) search for a more unknow hypothesis space, which may or may not be the right direction to go.

In terms of time complexity, for each iteration, RHC and SA are the fastest, followed by backpropagation and finally GA. This is because RHC/SA just try to find the next value by looking at neighbors, while back propagation needs to compute the gradient and attribute error. GA is the slowest in that because it needs to compute the fitness function for every member in the population, compute crossover for all mating pairs and then mutate the chosen individuals.

We can use gradient descent(backpropagation) whenever the fitness function/cost function is differentiable. Backpropagation usually has the best performance and need shortest time in this case. When the fitness function is not differentiable, RHC/SA/GA can be used. We will introduce other optimization problem in the section.

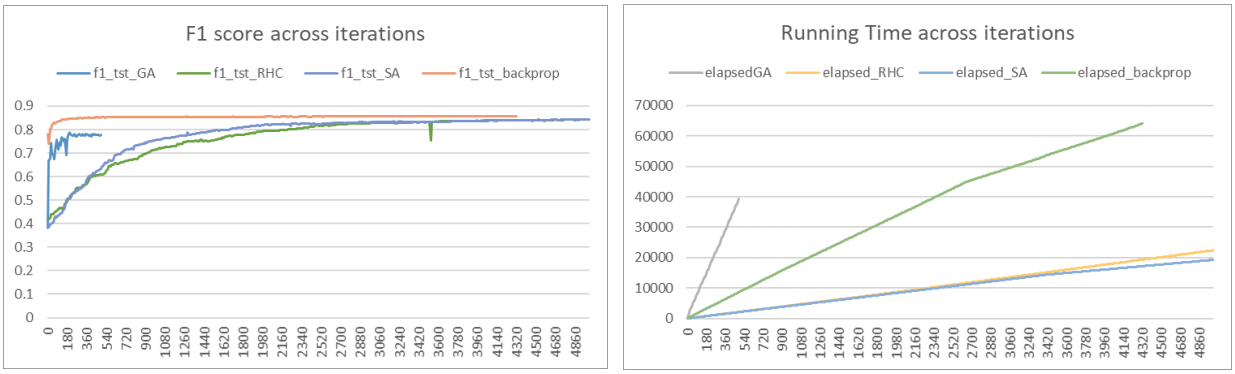

Figure 1.5 Analysis of four algorithms F1 score (left), running time (right)

2. Discrete Optimization problems

2.1 Traveling Sales Problem

Problem Statement and why it is interesting

Traveling Sales problem is defined as to optimize the trip for a sales person to visit n cities. The best solution is to make the trip as short as possible. It is a classic NP hard problem. The solution of the TSP problem can be represented by a n dimensional vector. However, we must make sure that one city is hit exactly once This problem has practical meaning for everyday lives. The problem is chosen to prove GA's ability to find best solution by crossover and mutation.

Analysis

Figure 2.1.1 shows the performance of the 4 algorithms on the Traveling Sales Problem (100 cities). GA has much better performance than other three algorithms. RHC/SA stuck at local optimal and thus cannot converge. MIMIC's one parent dependency tree assumption is not satisfied for the Traveling Sales Problem and thus had the worst performance among the 4 algorithms.

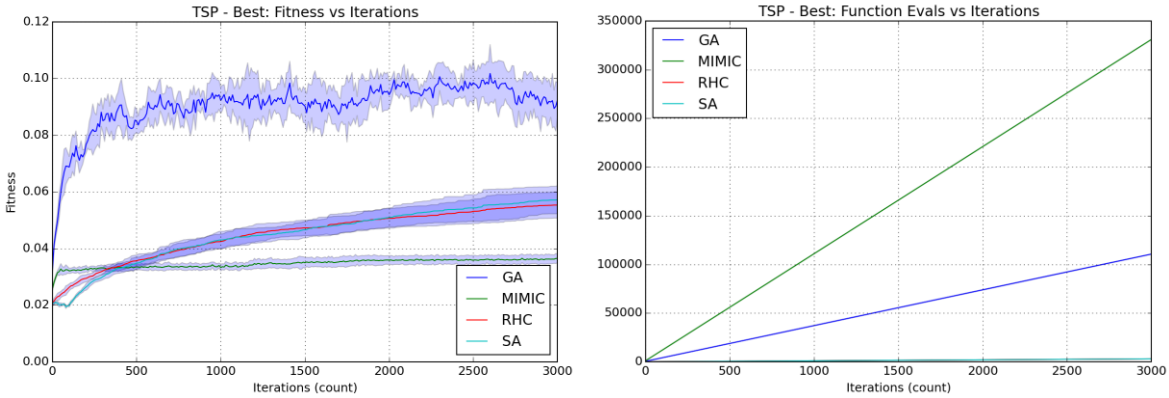

Figure 2.1.1 Empirical Result of four algorithms on Travel sales problem

Like we mentioned in the previous sections, population size, mate pairs and mutation will affect the performance of GA. In figure 2.1.2, we show different combinations of mutation and mate pairs with the population size of 100. Generally, the fitness function is maximized when mate pairs = 50, while mutate = 10(The blue lines shows on the right chart)

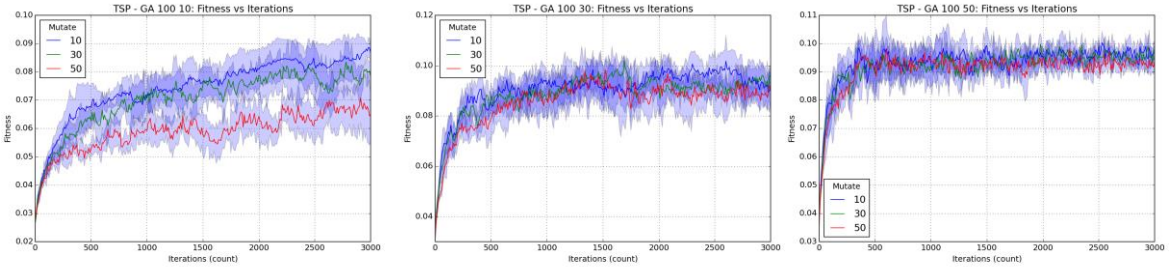

Figure 2.1.2 Mates/mutations estimate for the GA algorithm

TSP-GA mate=10(left), mate=30(middle), mate=50(right)

2.2 Continuous peaks

Problem statement and why it is interesting

Continuous peaks is an extension of the classic 4 peaks and 6 peaks problem. The problem is defined as, given a n dimensional input vector X as binary string, we want to optimize the following fitness function: $f(x,T) = \max(continue(0,x) + continue(1,x)\ ) + R(x,T)$

Where continue(b,x) is the number of bits of b in x, and R(x, T) is n when $continue(0,x)$>T and $continue(1,x)$>T, elsewise R(x, T) is 0. Popular algorithms like hill climbing can easily stuck at local optimal by make continue(0,x) = n or continue(1,x) =n. When T increases, the likelihood of getting stuck in the local optimal grows exponentially. This problem is chosen for SA to show its ability to jump out of local optimal.

Analysis

Figure 2.1.1 show the empirical result of optimization problem in GA, RHC, MIMC and SA when x is a 1000 dimension vector and T = 29. SA has the best performance in the continuous peaks and outperform all other algorithm. It is interesting that MIMIC works as the best at few hundred iterations. RHC is not doing well and had the lowest fitness in first 3000 iterations. GA is the second-best performing algorithm but take a significantly long time than other algorithms. The main reason that MIMIC is not working is that it assumes the dependency tree structure when generating the sampling distribution, which is not case for the continuous peaks problem. RHC stuck at the local optimal at before 4000 iterations.

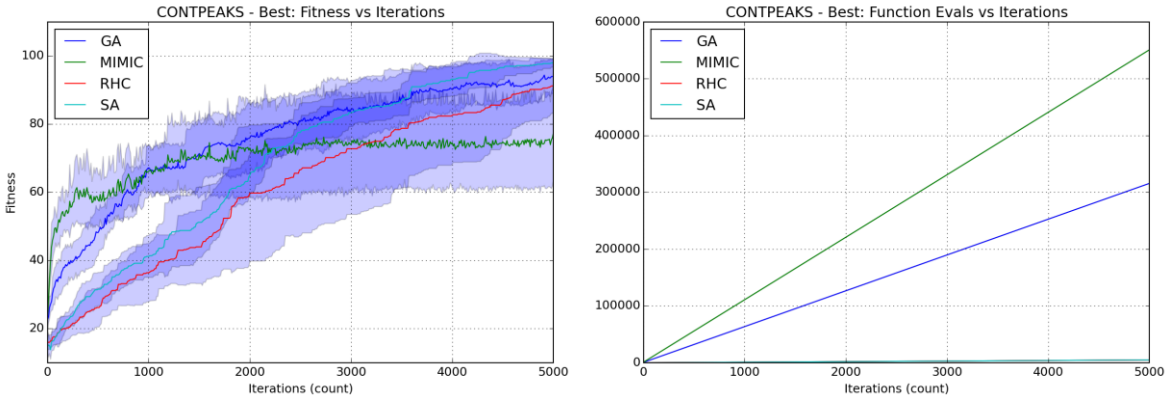

Figure 2.2.1 Empirical Result of four algorithms on Continuous Peaks

Figure 2.2.2 shows the temperature parameter tuning for SA, it is clear that when the temperature (CE) is low, the SA algorithm doesn't work well because it just behaves like Hill Climbing and suffer from the local optimal problem. Higher CE usually takes longer time because of the computation of CE. But generally give a better result.

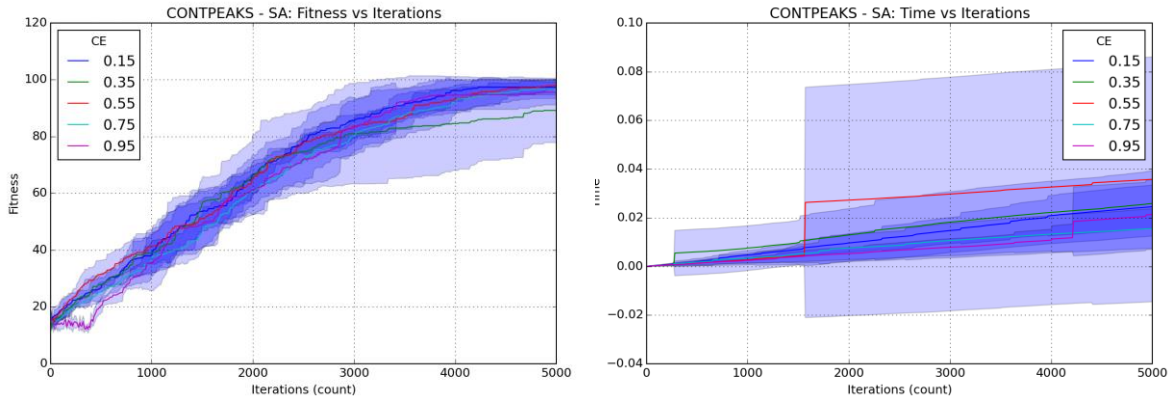

Figure 2.2.2 Temperature (CE) setting for the SA algorithm

2.3 Flip-flop

Problem statement and why it is interesting

The flip-flop optimization problem originally comes from the electrical engineer domain to design switches. It is defined as to optimize the following problem, given a binary string vector with x dimension, the fitness function is maximized when the pairs of continuous X_{t+1} != X_t is maximized. This problem is selected to prove the effective of MIMIC. Since any bit in the flip-flop bit string should be chosen by only looking at the previous bits. The chain rule is plotted in Figure 2.3.1:

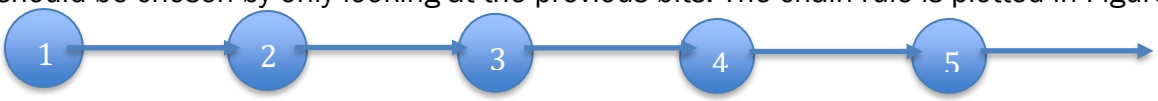

Figure 2.3.1 Chain rule of flip-flop problem

Analysis

The performance and running time of 4 optimization algorithms is shown in Figure 2.3.1 for a vector space = 1000. MIMIC works as the best algorithm across all iterations. It works wells because the chain rule of flip-flop (the choice of next bit in the string only depends on the previous bit) is aligned with MIMIC's assumption of dependency tree – every element only depends on one parent. SA and RHC has the same performance for the flipflop problem. GA is the worst of the 4 algorithms. All three algorithms other than MIMIC fail to converge after 5000 iterations.

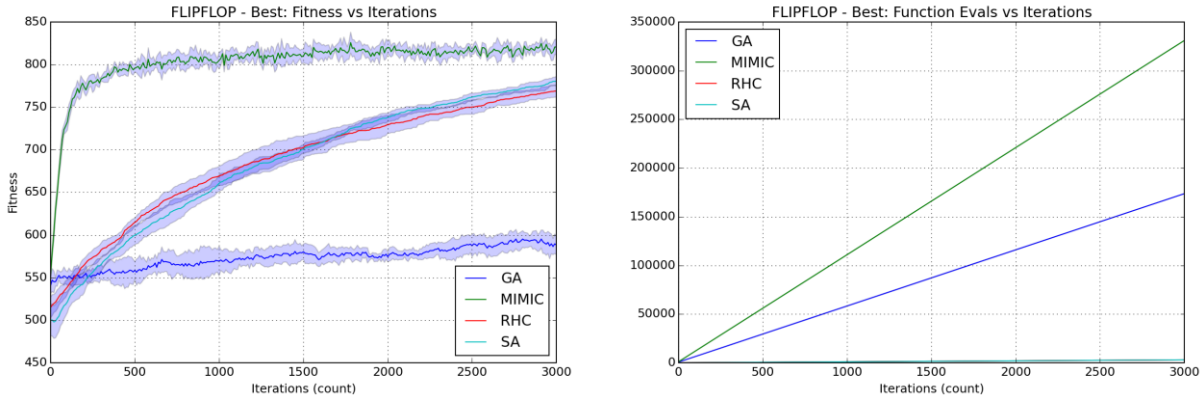

Figure 2.3.1 Empirical Result of four algorithms on Flip-flop problem

To optimize the MIMIC algorithm for the problem, we tune the parameter of m estimate. The parameter m is called equivalent sample size, which is used to solve the zero-count problem when estimating the probability in dependency tree. Without m, P(x=x1|parent) can be estimated by counts of parent divided by count of x=x1 with the given parent -- $N_{x1}/N_{parent.}$ After adding m P(x=x1|parent) = $(N_{x1}+mp)/(N_{parent.}+ m)$. A higher m usually means more smoothing when calculating the probabilities. Figure 2.3.2 shows that when M = 0.9, MIMIC has the best performance (with slightly more computation time).

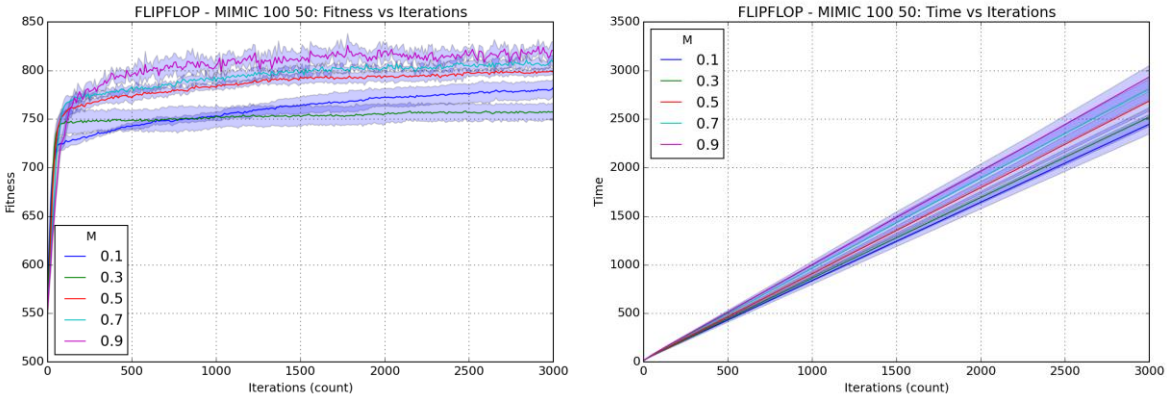

Figure 2.3.2 M estimate for the MIMIC algorithm

Conclusion

After exploring the 4 randomized optimization algorithms, I get a better understanding on how each of the algorithm works, the key parameters for each problem, and appropriate situation for each algorithm.

Back-propagation is still the best optimization algorithm for neutral network, beating all other three algorithms. This is mainly due to fact that we use MSE as the cost function for the neural network weights, which gradient can be easily computed. When the fitness functions are really complex and cannot be differentiable, SA/RHC/GA can be leveraged to find a good solution after sufficient iterations.

Besides the ones listed in the assignment, there are a lot of more optimization problems and optimization algorithms to explore. We always need to ensure the structure of the problem aligning with the algorithm structure (like the one parent assumption for MIMIC).