Abstract:

In this report, the performance of 4 randomized optimization algorithms (random hill climbing, simulated annealing, genetic algorithm and MIMIC) are investigated by two parts of work. For the first part, the first 3 algorithms are applied to to optimize the weights for neural network. For the second part, all the four algorithms are used to solve three optimization problems.

Environment:

Language: Java 8

Package used: ABAGAIL[1]

Part 1: Optimize Weights for Neural Network

Datasets: First data set is the ***Phishing Websites Data Set***[2] from UCI dataset. It consists of 11055 website instances which are label as a phishing website or not. Thus it is featured as a binary classification problem. Each instance has 30 numerical attributes such as *web age*, *DNS record, page record* etc. The dataset is split to 80%(training) and 20%(test). The application of the corresponding machine learning model could be used for various aspects in the field of cyber security.

Results of Back Propagation (BP)

Firstly, the BP method is used as a benchmark for comparison. To maintain the same neural network structure among different cases, back propagation method is implemented in the ABAGAIL. Meanwhile, it is used to tune the number of perceptron in the hidden layer for the NN. The input layer consists of 30 perceptrons and the output layer consists of 2 perceptrons, which are corresponding to the instance attributes and the outcomes. Perceptrons of the hidden layer are tuned from {2, 4, 8} as shown in Fig.1.

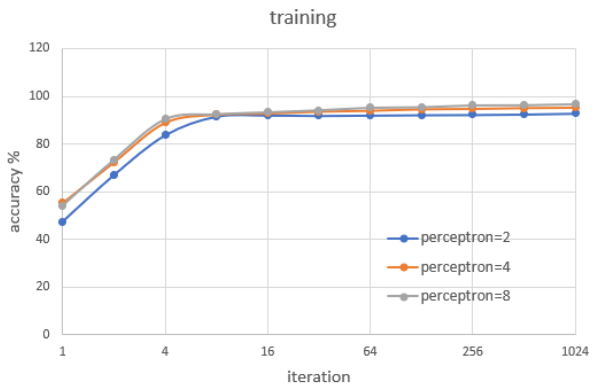

Fig. 1(a) Accuracy of training set

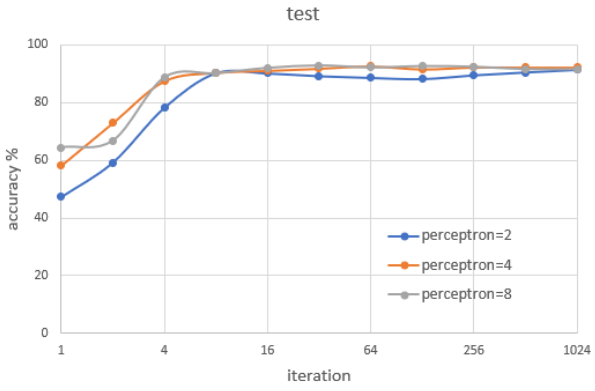

Fig. 1(b) Accuracy of test set

Fig. 1 Accuracy of back propagation using hidden layer with different perceptrons

For different number of perceptron, all cases yield high accuracy of prediction both on training set and test data. The difference are basically neglectable. Besides, all cases converge in a few iterations, around 16. Results shows that backpropagation method is pretty good at finding the weight for NN.

Fig. 2 Training time using hidden layer with different perceptrons

Fig.2 shows the training time for different cases. Basically the training time doubled while number of perceptron doubled. However the accuracy are at same level as discussed before. To maintain a simple NN structure avoiding overfitting, we applied hidden layer with two perceptrons. Meanwhile to make fair comparison among different cases and NN structure (30, 2, 2) will be used hereafter.

Results of Random Hill Climbing (RHC)

Fig.3 shows the accuracy on training set and test set. Result shows that accuracy end with a pretty high level (90%) at around 256 iteration. At the first several iterations (16), the accuracy doesn't change. It might be the reason for the random search. Thus the optimization is highly based on the number of iteration. Finally, it takes around 256 iteration to converge, which is much more than BP.

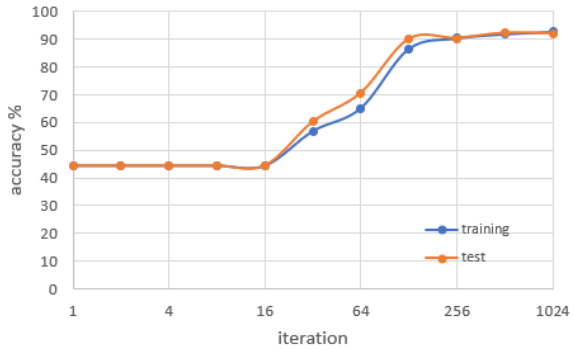

Fig. 3 Accuracy using RHC

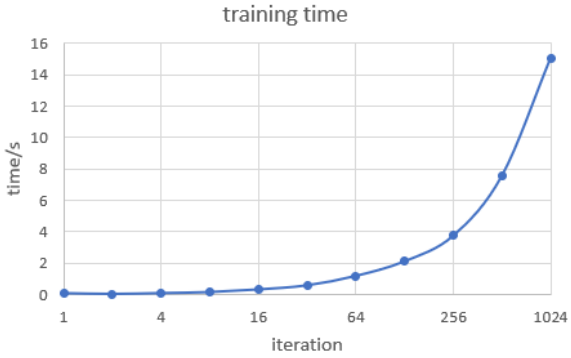

Fig. 4 Training time using RHC

Fig.4 shows the training time of different iteration. Though the NN structure are same, but RHC takes around 15s to run 1024 iteration, however BP took only around 6s.

Results of Simulated Annealing (SA)

For simulated annealing algorithm, there are two parameters controlling the optimization process, start temperature t and cooling exponent C. The start temperature decides the initial status of annealing and the cooling exponent decides the rate of cooling down. A lower cooling exponent results in higher rate of temperature decrease. Firstly, we tuned the cooling exponent (0.01, 0.1, 1) with constant start temperature 1e11.

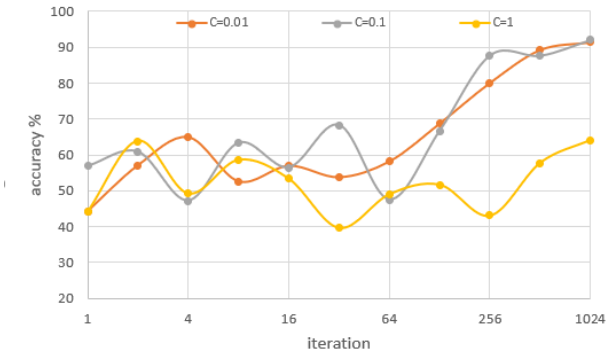

Fig. 5 Training accuracy with different C

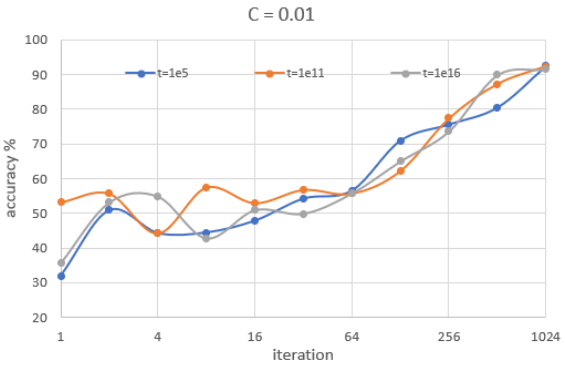

Fig. 6 Training accuracy with different t

As shown in Fig.5, the accuracy lines of all cases fluctuate, which dues to the feature “go down the hill” of SA. When the cooling exponent is relatively low (C=0.01), the fluctuation occurs only at first several iterations, because the temperature will be low after a few steps. So it is more like a hill climbing algorithm. When the cooling exponent is at a middle range (C=0.1), the accuracy goes up with the iteration with more fluctuations comparing to the case with C= 0.01. With a higher C, the temperature will be higher, thus the algorithm is more willing to go down the hill, which results in more fluctuations. However, both cases reach the highest accuracy with sufficient iteration (around 1024). For the case C=1, that means the temperature maintains same for all iteration. With a high temperature for all iterations, the accuracy fluctuates at a low level because the algorithm is willing to go down. Thus it is more like random walk.

Fig. 7 Test accuracy with different C

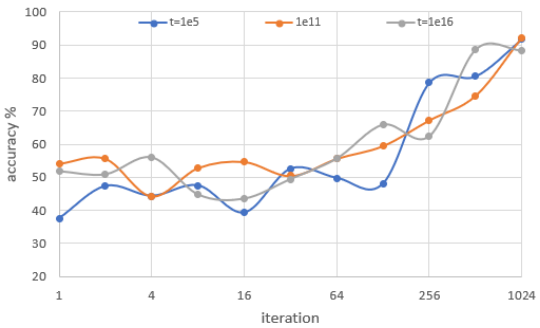

Fig. 8 Test accuracy with different t

Finally, we choose different start temperature (1e5, 1e11, 1e15) with same cooling exponent (0.01). As discussed before, the temperature will go down within several iteration for all start temperature, since the cooling exponent is low and the temperature decreases exponentially with cooling exponent. Thus the accuracy line of different cases are pretty similar except for first few iterations only. All cases reaches high accuracy after around 1024 iterations.

For test data, results are similar to training data, shown in Fig.7 and Fig.8. For the computing time, each iteration of SA is independent of parameters selected. 1024 iterations took around 15s. Detailed information will be showed in the comparison part later. To get a relatively stable results, we choose cooling exponent (C=0.01) and start temperature(1e11) for later comparison in part 1.

Results of Genetic Algorithm (GA)

For genetic algorithm, there are 3 parameters to tune, the population size (Ps), the mate rate (crossover rate) and the mutation rate (Rm). Previous studies showed that usually the crossover rate (Rc) ranges from 0.6-0.9, whereas Rm ranges from (0.1-0.4) [3]. Thus we investigate the population size (in the range {10, 40, 160}) firstly using a fix Rc=0.8 and Rm=0.2.

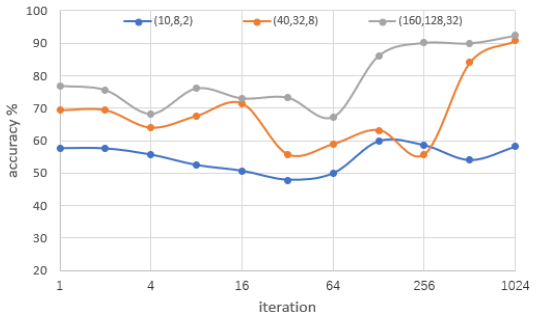

Fig. 9 Training accuracy with different Ps

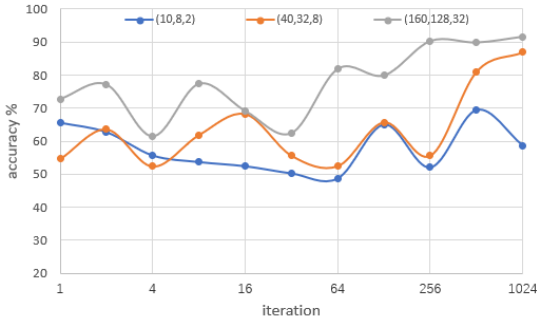

Fig. 10 Test accuracy with different Ps

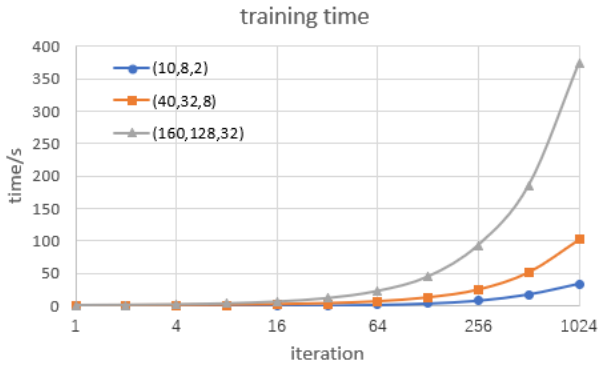

Fig. 11 Training accuracy with different Ps

Fig.7 and Fig.8 show the training accuracy and test accuracy using different population size, mate individuals and mutate individuals. With same crossover rate and mutation rate, larger population size results in higher accuracy. However both (10,8,2) and (40,32,2) cases yield 90% accuracy with 1024 iterations. The fluctuation is caused by a relatively high mutation rate.

Fig.9 shows the training time of different cases. The time cost increases linearly with the population size. To keep a simple structure of algorithm avoiding overfitting and save time, population size of 40 will be used to tune the crossover rate Rc and mutation rate Rm. Considering the dependency of Rc and Rm, we compared four cases: (Ps=400, Rc=0.6, Rm=0.1), (Ps=400, Rc=0.8, Rm=0.1), (Ps=400, Rc=0.6, Rm=0.2) and (Ps=400, Rc=0.8, Rm=0.2).

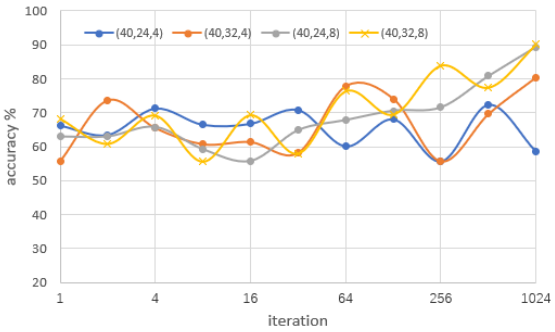

Fig. 12 Training accuracy

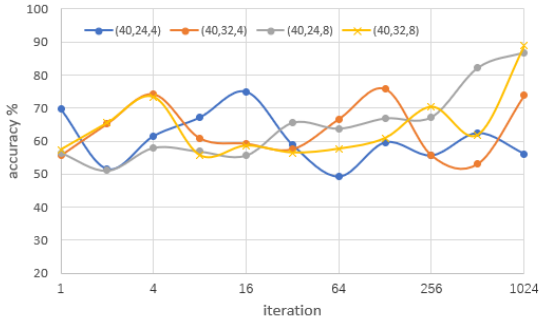

Fig. 13 Test accuracy

As shown in Fig.12 and Fig.13, case(40,24,8) with Rc=0.6 and Rm=0.1 shows relatively stable and high accuracy. Thus it will be used for later comparison.

Algorithms comparison

As shown in Fig.13, Fig.14 and Fig.15, back propagation method shows the best performance. It coverages within few iterations and cost least time for each iteration. For the other three algorithms, basically all of them could reach high accuracy with enough iterations. However, GA takes much more time than RHC and SA. For GA and SA, both of them show fluctuation which is caused by the “go down the hill” feature of those methods.

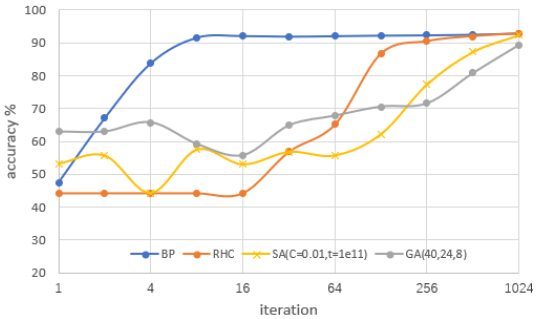

Fig. 14 Training accuracy

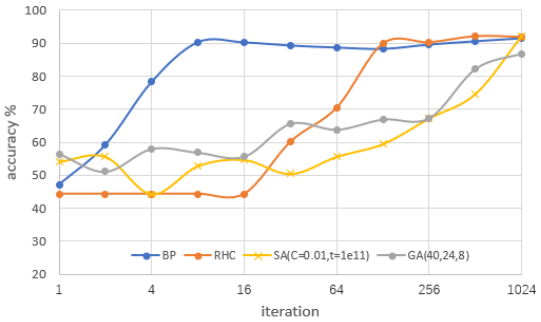

Fig. 15 Test accuracy

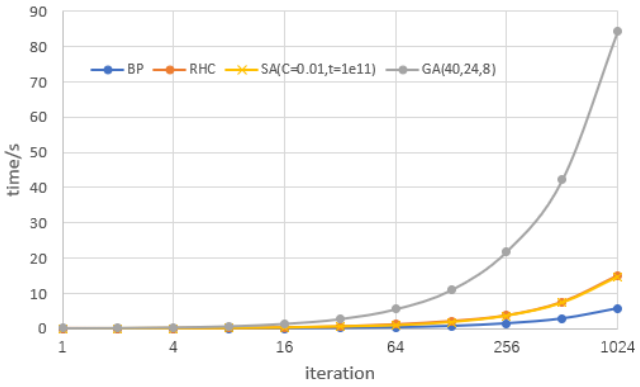

Fig. 16 Training time of different algorithms

Part 2: Optimization Problems

In this section, we will apply RHC, SA, GA and MIMIC to three optimization problems, Continuous peak, travelling salesman and Knapsack. Each problem will highlight an algorithm of SA, GA or MIMIC. To account for the randomness, each algorithm will be run 3 times to take an average for the fit function and computing time.

As discussed in part 1, parameters selection may influence the optimization result, so it is hard to make a fair comparison among those algorithms. However, we will use the same parameters for all three optimization problems to check their relative performance on different cases.

For RHC, no parameter needs to be defined in ABAGAIL.

For SA, the cooling exponent C=0.8 is used to keep the temperature and prevent it degenerate to Hill Climbing Algorithm, and the start temperature t=1e11 is used same as in part 1.

For GA, population size=40, mate rate=0.6 and mutation rate=0.2 are used, which is corresponding to the best case for GA in part 1, which is shown as (40, 24, 8).

For MIMIC, sample size=40 is used to make a relatively fair comparison to GA, and the top 50% individuals will be kept, which is shown as (40,20).

To define the best algorithm for each problem, we will consider the optimized fitness over 1024 iteration firstly and the time cost of 1024 iterations secondly.

Continuous Peaks

Continuous peak problem (CPP) is to find the highest peak of the domain (global optimum) among all the peaks (local optima). It is quite a simple optimization problem, but to escape from the local optima is a general difficulty facing by various optimization problems. Thus the CPP could highlight algorithms' ability of identifying local solution, or "go down the hill". In this report, N=50 and T=5 are used in ABAGAIL.

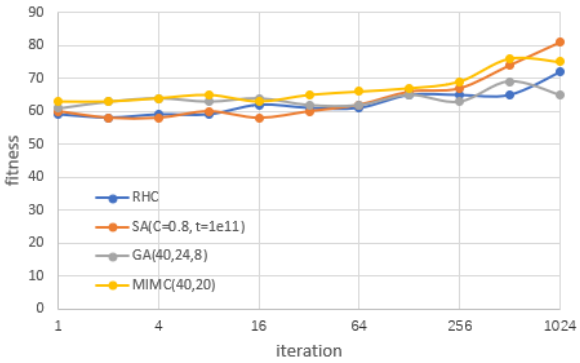

Fig. 17 Fitness optimization

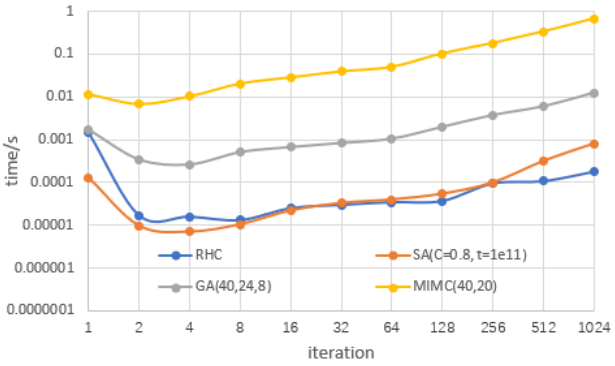

Fig. 18 Time cost

Shown as Fig.17 because it is a relatively simple problem, all algorithm could yield good results given 1024 iteration. SA gives the highest fitness around 80 with 1024 iteration. The other is around 70. All the fitness curves show some fluctuations, which are resulted from the "go down the hill" feature of those randomized optimization algorithms.

Fig.19 shows that SA and RHC are two fastest algorithms and are at the same order basically. Whereas GA costs more time than SA and RHC but gives lowest accuracy. For MIMIC, it could gives good fitness optimization, however the time used is much more than the others. It could be the reason that MIMIC is good with a complex fitness function but each iteration will be expensive to use. For a simple problem like CPP, MIMIC

Notice that time cost first several iteration goes down, it may due to the initialization process. All the other problems have similar phenomena, we will not explain it hereafter.

In all, simulated annealing outperforms all the other algorithms in this problem.

Travelling salesman

The travelling salesman problem (TSP) is to find shortest route of a list of cities and returns to original one. It is featured as a NP-hard problem in combinatorial optimization. The worst-case running time of TSP problem increases at least superpolynomially with the number of cities N. In this report, N=50 is used.

Fig.17 and Fig.18 show the fitness optimization process and time cost for different algorithms. For a given iteration, GA is clearly the best method among all those 4 algorithms. The fitness of GA is much higher than RHC, SA and MIMIC at all iterations. For the computing time, MIMIC costs much more than the others. RHC and SSA are the least and basically same. GA is at the middle.

Balance the fitness optimization and time cost, we believe GA is the best algorithm for this problem that gives highest fitness value with a medium cost of time.

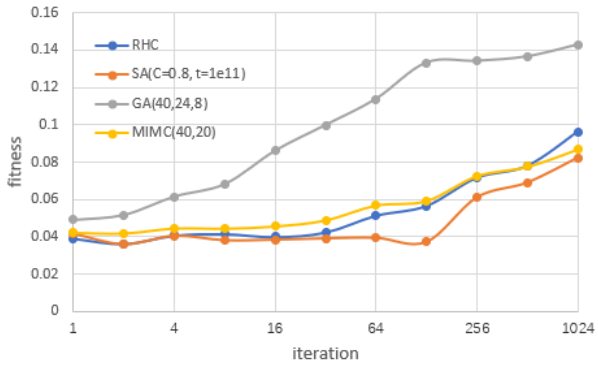

Fig. 19 Fitness optimization

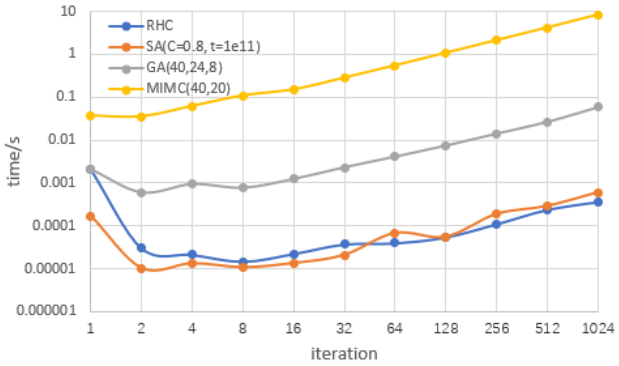

Fig. 20 Time cost

Knapsack problem

The knapsack problem is also a NP hard problem for combinatorial optimization: Given a set of items that are of a weight and a value, select items to give the largest value while meets the limitation of maximum weight. The problem is wildly studied in fields related to resource allocation such as financial investment and scientific computing.

Shown as Fig.21, at all iterations, MIMIC outperforms the other algorithms. After 1024 iterations, RHC, SA and GA reach a fitness around 3000, whereas MIMIC is remarkably above 3500.

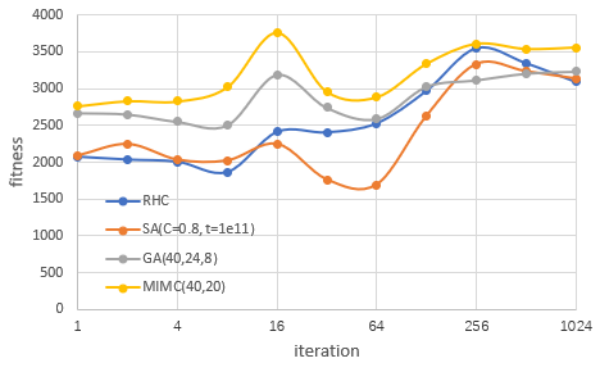

Fig. 21 Fitness optimization

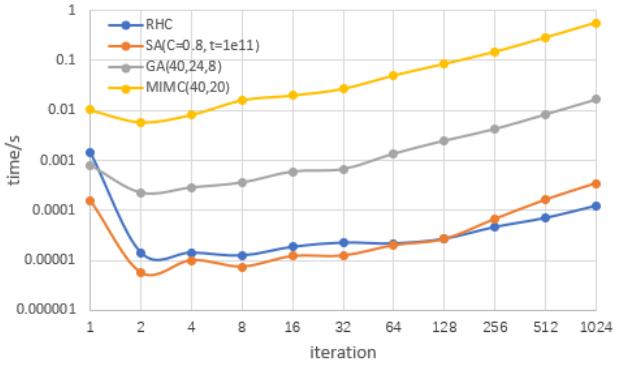

Fig. 22 Time cost

Once again, for same amount of iteration, RHC and SA cost least time and GA costs more than RHC/GA but less than MIMIC, as shown in Fig.22. But considering that MIMIC reach a high fitness at 16 iteration whereas all the other algorithms never reach it within 1024 iterations.

Based on our criteria, the MIMIC is the best algorithm for this problem.

Algorithm Comparison

In this section, we applied RHC, SA, GA and MIMIC to three different optimization problems. Each algorithm has its own advantage on particular problem, based on the problem structure or fitness function used. Detailed remarks could be shown as Tab.1

Tab. 1 Algorithm Comparison

Algorithms	Parameters	Time cost	Suitable problem
RHC	0	Low	Simple structure, simple fitness function
SA	2	Low	Simple structure, simple fitness function
GA	3	Medium	Complex structure, simple fitness function
MIMIC	2	High	Complex structure, Complex fitness function

As we discussed before, it is hard to make fair comparison among those algorithms with various parameters. However we could use algorithms with constant parameters over three optimization problems to study the algorithms' relative performance for different kinds of problems.

For the continuous peaks problem, it has a simple structure and a simple fitness function. SA yields highest fitness with relatively short time. However for the travelling salesman problem and knapsack problem, they are featured as NP hard problem with complex structure. GA and MIMIC outperform other methods in these two respectively. Thus we conclude that RHC and SA are suitable for simple structure and simple fitness function problems. GA can handle the complex structure, however it may need many iterations to converge, so a simple fitness function should be better for GA. Finally, MIMIC is good a the problem with complex structure and fitness function.

Summary:

In this report, we applied several randomize algorithms to different optimization problem. In the part 1, we used back propagation, RHC, SA, GA to optimize the weights for neural network. In the part 2, we adopted RHC, SA, GA and MIMIC to solve three optimization problem. Detail performance could be concluded as:

Back Propagation: It is the best algorithm for tuning the weights for NN. The accuracy is the highest and the time cost is the lowest. Because in this case, we could solve the direction of changing the weights, so we do not need to randomly search the solution domain.

Random Hill Climbing: For tuning the weights, it shows good performance comparing to SA and GA. The time costed is at same level as SA, but much shorter than GA. For optimization problems, RHC shows acceptable performance. Though the time cost is low, it hardly can reach the accuracy as the best algorithm.

Simulated Annealing: SA has two parameters to adjust the optimization, the start temperature and cooling exponent. With a low cooling exponent, the temperature will decrease fast and therefore the influence of start temperature will be small. For tuning the weights, SA yields acceptable results with a low level of time cost. For optimization problems, SA shows good performance for simple structure problem such as continuous peak problem.

Genetic Algorithm: GA has three parameters, population size, mate rate and mutation rate. After tuning the parameters, GA yields acceptable results for optimizing the weights. However the time cost is more than RHC and SA. For optimization problems, RHC shows good performance for travelling salesman problem, which has a complex structure.

MIMIC: It has two parameters, the sample size and percentile of kept individuals. MIMIC shows good performance on complex structure problem. Though the time cost for MIMIC is much more than RHC, SA and GA, it outperformed all the other randomize optimization algorithms on knapsack problem.

Reference:

1. https://github.com/pushkar/ABAGAIL

2. http://archive.ics.uci.edu/ml/datasets/Phishing+Websites

3. C. Srinivas, B. Ramgopal Reddy, K. Ramji, R. Naveen, Sensitivity Analysis to Determine the Parameters of Genetic Algorithm for Machine Layout, Procedia Materials Science, Volume 6, 2014, Pages 866-876,