Report for CS7641 Project 2: Randomized Optimization

Yiwei Yan

October 26, 2015

Abstract

Optimization algorithms are typically used in problems that have many possible solutions across many variables, and that have outcomes changing greatly depending on the combinations of these variables[1]. This report aims to explore random search and provide performance analysis of various optimization algorithms on neural network and three other optimization problems. The optimization algorithms include Randomized Hill Climb (RHC), Simulated Annealing (SA), Genetic Algorithms (GA), and Mutual-Information-Maximizing Input Clustering (MIMIC). This report first provides a discussion of using the first three algorithms to find good weights for a neural network based on iris dataset; then utilizes the four algorithms on three optimization problem domains, which should highlight advantages of genetic algorithm, simulated annealing, and MIMIC respectively. The langurage used for implementing all algorithms is Python with PyBrain, Mimicry, Matplotlib packages.

1 Methodologies Introduction

Optimization is mostly utilized in cases when there are too many possible solutions exist. Optimization explores the best solution to a problem by scoring them (e.g. minimize cost function) to determine quality. Cost function is usually the key for solving problem using optimization methods, but it is usually the most difficult part to define. The goal is to try to minimize the cost function and let it return a value that can represent the quality of solutions (smaller values for better solutions).

1.1 Randomized Hill Climb (RHC)

RHC can be understood as a greedy algorithm. It starts at a random solution and relace it with a searched nearby better solution that has a lower cost. This process iteratively goes on until there is none of the neighboring solutions can decrese the cost. Although RHC can quickly convergence to an good solution, the solution may only be an local optimal solution rather than a global optimal solution. An example of suitable problem for RHC is a problem has only one optima (local optima is the same as global optima).

1.2 Simulated Annealing (SA)

Annealing is a physics process of heating up an alloy and then cooling it down slowly. It uses a variable representing the temperature, which starts very high and gradually gets lower. In each iteration, one of the numbers in the solution is randomly chosen and changed in a certain direction[1].

Simulated annealing is much like the RHC method, but it adds in random factors for its search process. Except for accepting better solutions, simulated annealing can also accept worse solution with a certain probability. The probability of a higher-cost solution being accepted is given by the following formula: $P(e, e', T) = exp(-(e' - e)/T)), \forall e' < e$. Therefore, it may attempt to jump out of the local minimum problem and have a chance to reach the global optimal solution. Compared to RHC, SA is more suitable for the problems with local optima, but the disadvantage of SA is that it have more randomized effect which may result in longer search time. In the extreme case when T is extremely large, SA degenerates to random walks.

1.3 Genetic Algorithms (GA)

Genetic algorithms is another optimization algorithm that is inspired by a natural selection process which mimics biological evolution.

GA initially creates a population of random individual chromosomes (solutions). At each step of optimization, the algorithm repeatedly modifies the population by randomly selecting individuals from the current population and uses them to generate the next generation. Basically, individuals that fit better in each competition will produce more offspring than those individuals that perform poorly. Genes from good individuals propagate throughout the population. Therefore, after successive generations, the population may get an optimal solution.

The algorithm evolves three main processes:

1. Selection which equates to survival of the fittest;
2. Crossover which represents mating between individuals;
3. Mutation which introduces random modifications.

The advantage of GA is that it has a strong global search capability. The disadvantage is its running time is usually longer and the convergence process is slower, and it is easy to be affected by different parameters. GA can be applied to problems like objective function is discontinuous, non-differentiable, stochastic, or highly nonlinear.

1.4 Mutual-Information-Maximizing Input Clustering (MIMIC)

The basic idea of MIMIC algorithm is to create a process of successive approximation[2]. Firstly, it generates a random uniformly selected population of individuals from the input space. From this population, the median fitness is extracted as θ_0. Then, give a collection of points that `cost_function(x)` is not larger than the threshold θ_0. Based on those points, it constructs a density estimator for $P^{\theta_0}(x)$. This density estimator can generate additional samples and establish a new threshold $\theta_1 = \theta_0 - \epsilon$ to be used to construct another new density estimator. This process repeatedly goes on until the values of `cost_function(x)` stop improving. The performance of this approach is dependent on the nature of the density approximator used.

The advantage of this algorithm is that though the running and convergence speed is slow, but it can find better solutions than other optimization algorithms[2].

2 Optimize Neural Network(NN) Using Randomize Optimization Algorithms

In the analysis of our last report, we generally applied back-propagation training method to optimize NN weights on different datasets. In this section, we utilize RHC, SA, GA to optimize the NN weights; then present the experimental results and comparison of back-propagation, RHC, SA, GA applied on an example Neural Network model on the iris dataset. In our last report, we obtained classification accuracy around 0.95. Thus, in this experiment, we aim to validate the accuracies from RHC, SA and GA, and investigate what parameters affect their results. To improve the performance of the investigated algorithms, we perform *parameter sweeps* on the key controlling parameters.

2.1 Backpropagation

Iris dataset has 150 samples with four-dimensional features in three classes. The structure of NN we use for iris dataset is with four input nodes, one hidden layer with three nodes (Sigmoid), and one output layer with three nodes (SoftMax). 70% of shuffled iris data is as training data, while the other 30% is used as testing data.

Table 1: Training time comparison of RHC, SA, GA on Iris NN.

Algorithm	**Iteration**	**Training Time (s)**
BackProb	500	29.204
RHC	500	4.635
SA	500	4.697
GA	500	47.132

For benchmarking, we first apply the back-propagation to train a NN on Iris dataset, and validate the accuracy over the iterations. As shown in the figure 1 subfigure 1 in the period of 500 iterations, the training and testing errors of BP optimization become stable around the range [3%, 8%] after 200 iterations, which means it best accuracy is around 95%.

2.2 Comparison among RHC, SA, and GA

In terms of RHC, GA, and SA optimization algorithms, their training and testing errors shown in figure 1 and the timing in table 1.

RHC algorithm gives a relatively smooth decreasing trend for both training and testing error. After 450 iterations, RHS gradually stabilizes to training error at around 0.05 and testing error at about 0.07. RHC's training is very efficient, which takes only 4.635 seconds for 500 iterations. On the other hand, the accuracy of SA fluctuates over the iterations, in which the training and testing errors seem increase from the beginning to the 400th iteration, and then decrease after that, but increase again at around 500th iteration. At the best performance, its testing error is around 0.055 and the training error around 0.13 (higher than the testing error). From this process we can clearly see that SA more randomized effect than RHC. Its training time is very similar to RHC, because the atom operation in each iteration is very similar to RHC (though it needs to evaluate the probability $P(e, e', T)$, this operation is every efficient). Both RHC and SA are very fast for training, and have low training and testing errors in general.

The performance of GA is very good, which give lower training and testing accuracies compared to RHC and SA. The testing error decreases to about 0.02 after 300 iterations. Moreover, the training error curve is smoother than RHC and SA, though the testing error fluctuates (more in the first 350 iterations). However, GA costs a long time on training compared to RHC and SA, as listed in table 1.

After performing parameter sweeps, we find that the performances of those optimization methods can be improved by changing their parameter values.

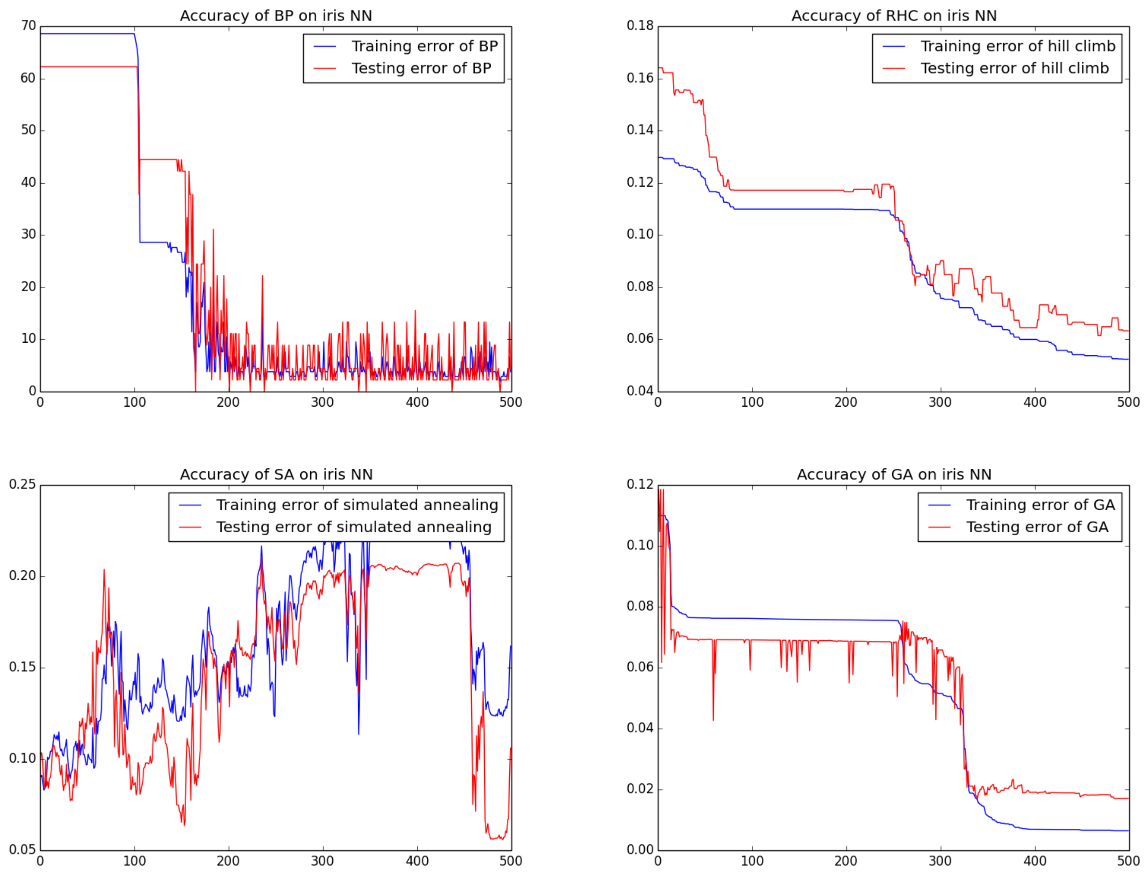
For RHC, as shown in figure 1, its performance can be improved by increasing the number of iterations. For RHC, with random restarts it can randomly pick a new starting point that can be helpful for it to jump out of a local optima and have a chance to reach the global optima.

For SA, the iteration times for its performance is not as helpful as for RHC, because the randomized "step-back" operation can possibly change the parameter to a wrong direction in the additional iteration. The important controlling parameters of SA are the SA `temperature` and the `max-learning-step`. On the Iris dataset, when decreasing the temperature of SA from 10000 to 0.1, it causes the training and testing errors to go up, but training and testing time go down. ON the other hand, when we decrease max-learning-step of SA from 1000 to 0.5, the training and testing errors become larger while the training time goes down, as depicted in figure 2. Therefore, we could image that the performance of SA can be increased by choosing a *reasonably* large temperature and a reasonably low learning step. In our training for NN on iris data in figure 1, we use $learningstep = 1$, and $temperature = 1000$.

For GA, the important controlling parameters are its `mutation percentage` and `elite percentage` (the percentage of top proportion samples).

As depicted in figure 3, when we decrease the mutation percentage from 1 to 0.01, we can clearly see that the training error and testing error go down, so do the training time. The same

Figure 1: Training and testing errors of BP, RHC, SA, GA on Iris NN over 500 iterations.



situations happen on GA when we decrease the percentage of top proportion.

Therefore, it can be expected that the performance of GA can be increased by choosing a reasonable low mutation and top proportion percentage. In our training for NN on iris data in figure 1, we use `mutation percentage` = 0.2, and `elite percentage` = 0.2.

3 Optimization Problem 1: Group Travel

3.1 Problem Statement

The first optimization problem is about planning a lowest-cost flight trip for a group of people that from different locations to meet togehter and travel to the same destination. This case is interesting because it is a discrete optimization problem with discontinuous, non-differentiable, stochastic, or highly nonlinear objective function.

Genetic algorithm has a good capability to solve such ”time arrangement” problems. We try to highlight the advantage of GA from other methodologies with this problem.

The dataset we utilize for this problem is named as `schedule.txt`. There are 5 columns in this dataset, which represent original location, destination location, departure time, arrival time and price respectively. For example:

There are in total 6 people in the travel group. Every person needs to choose two flights to take (outbound and return). There are ten flights each person can choose from where 0 means the first flight of the day, and 1 means the second, etc. A example of flight combination of the group of people is $[1, 4, 3, 2, 7, 3, 6, 3, 2, 4, 5, 3]$. For instance, suppose the first person is Mary, then 1 represents her outbound flight and 6 represents her return flight.

The goal is to find a set of inputs (flights) that minimizes the total cost of travel. There are

Figure 2: NN-SA: Accuracy Change by Decreasing Temperature & Learning Step.

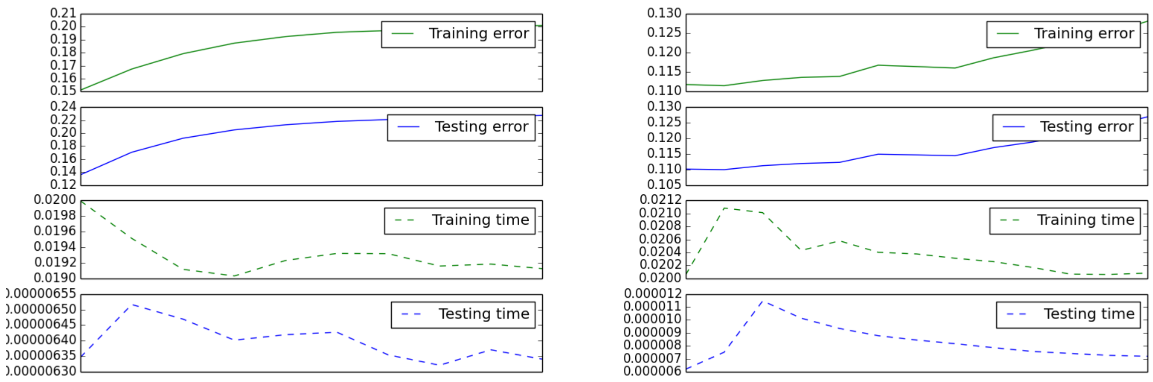
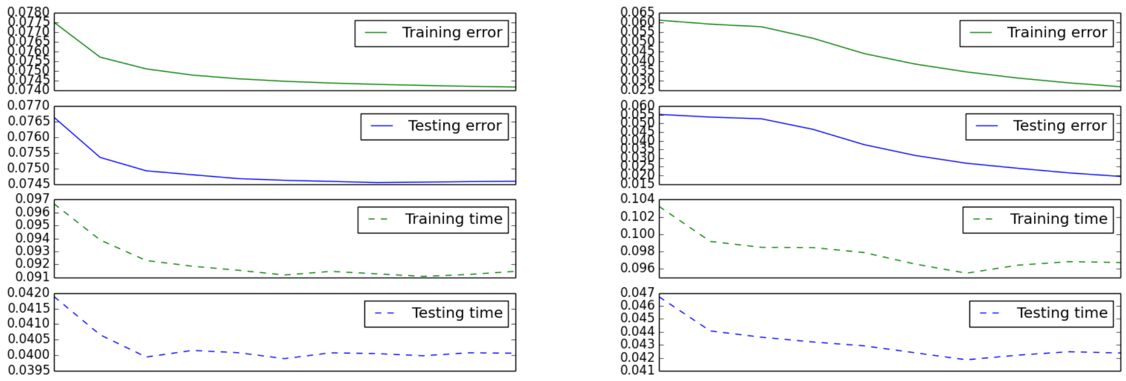


Figure 3: NN-GA: Accuracy Change by Decreasing Mutation $ Elite Percentage.



some important impact factors that are considered in our measure of cost function:

(1) Total cost of the trip;

(2) Total time spent for waiting others to arrive;

(3) 50 dollar penalty for the rental car one-day-late fee.

The domain is therefore

$$domain1 = [(0, 9)] \times (6 \times 2). \tag{1}$$

3.2 Results from Optimization Algorithms

Since the randomized optimization's performance may roughly change after restarting each time, we set 20 iterations and collect the relatively stable results from those algorithms.

(1) The optimized result from RHC is : $[8, 5, 3, 7, 1, 5, 8, 6, 8, 5, 2, 6]$, with total cost \$3017.

(2) The optimized result from SA is: $[7, 3, 5, 2, 1, 5, 6, 9, 2, 5, 6, 3]$, with total cost \$3096.

(3) The optimized result from GA is: $[4, 5, 3, 5, 4, 5, 3, 6, 4, 5, 4, 6]$, with total cost \$2802.

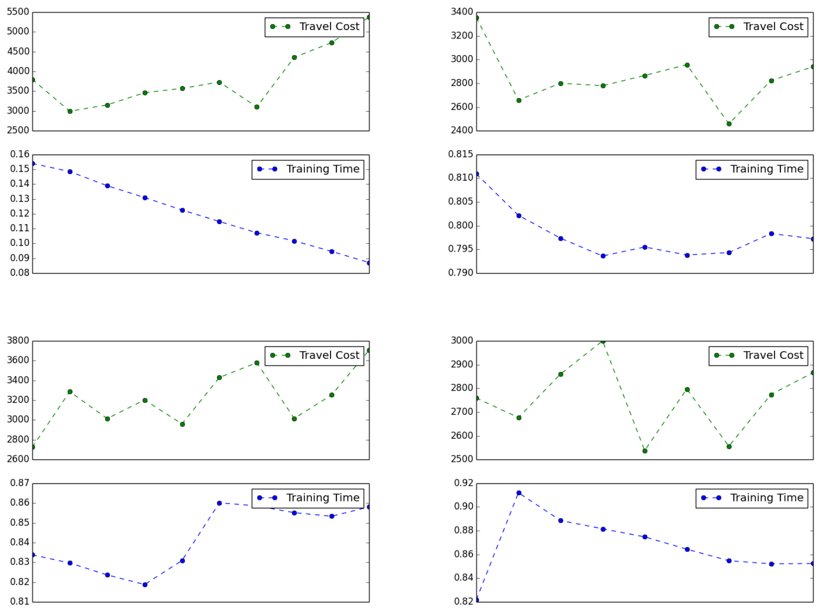
(4) The optimized result from MIMIC is: $[7, 1, 3, 2, 6, 1, 6, 1, 6, 1, 4, 1]$, with total cost \$2988.

As the results summarized in table 2: the fastest algorithms to find the lowest cost travel plan are RHC and SA, both of which finish in less than 6 seconds. The optimized costs are \$3017 and \$3096 respectively. Compared to this, GA with the training time 59.5 seconds is around 10 times slower than RHC and SA, but it gets a better low-cost result of \$2802. In terms of MIMIC, it gets a similar good result (\$2988) to GA , but it is extremely slow which is around 13 times slower than GA. According to the performance comparison, we may want to use GA rather than the other algorithms in this travel arrangement problem.

Table 2: Optimization Problem 1: Result Comparison After Optimization.

Algorithm	Iteration	Total Cost	Training Time (s)
RHC	20	3017	5.697
SA	20	3096	4.741
GA	20	2802	59.483
MIMIC	20	2988	772.66

Figure 4: Optimization Problem 1 with GA: Value Trends by Changing Pop-size (top-left), Mutate-prob (top-right), Learn-step (bottom-left), Elite-prob(bottom-right).



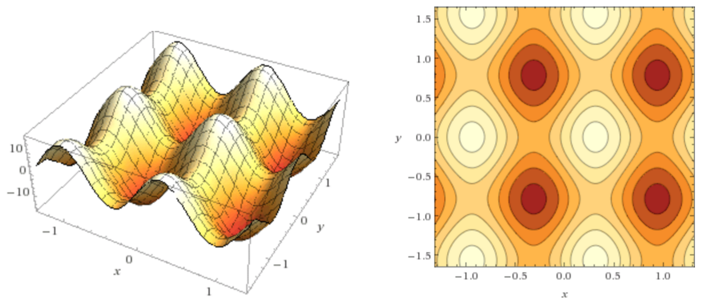
3.3 Improving GA Performance

In the original model, we use 10 as population size, 1 as learning steps, 0.2 as mutation percent, and 0.2 as elite percentage. To further improve the performance, we perform parameter sweeps to select better parameter values.

In figure 4, we plot the travel-cost and training time when decreasing value of the important parameters, including population size, mutation percent, learning steps, and elite percent.

When we decrease population size of GA from 10 to 1, we found that it was good to pick 9 as the population size based on the lowest value of travel cost even though the training time was relatively longer. For mutation percentage from 0.9 to 0.1 with interval 0.1, it can be seen that 0.3 is a good choice considering both training time and optimization performance. We can see that at the point of 0.2 from figure 5, the travel cost value is the lowest. Besides, for learning steps, we tested from 1 to 0.1 with the interval 0.1. It can be seen that 1 is the best choice because of lower training time and lowest travel cost. Lastly, for elite rate from 0.9 to 0.1 with the interval 0.1, we can see that a value between 0.15 to 0.35 gives good performance considering both training time and low travel cost. Thus, our original choice 0.2 is still good.

Figure 5: Optimization Problem 2: Cost Function.



4 Optimization Problem 2: Minimizing Function with Multiple Local Optima

4.1 Problem Statement

In the second optimization problem, we need choose two integers x and y between $[-50, 50]$ to find the minimum value of function:

$$f(x, y) = 10 \times \sin(5x) + 7 \times \cos(4y) + 17 \tag{2}$$

which can be visualized in 3D and 2D as in figure 5. In the continuous domain, the minimum value is 0.

This problem is interesting because, as shown in the above figures, there are a lot of local minimum values in this function even from a range $[-1, 1]$ (there is one local minimum value in each period defined by the sin and cos functions). Note that although in the continuous domain all the local minimum values are also the global minimum values, but in the **integer (discrete) domain** they may not be, which makes this problem interesting.

As discussed before, optimization algorithms may return a local minimum instead of a global minimum. It is easy for RHC and SA to be stuck in a local minimum value. Therefore, the goal of this case is to see that RHC and SA are not so efficient on finding global minimum value, while GA and MIMIC can overcome this deficiency sometimes. Moreover, the value obtained from MIMIC is better than GA.

Domain is therefore defined as:

$$domain2 = [(-50, 50)] \times 2. \tag{3}$$

4.2 Results from Optimization Algorithms:

Table 3 summarizes the results after running RHC, SA, GA, and MIMIC algorithms on operation problem 2. We use 10 times iterations to get the relatively stable results.

The fastest algorithm finding the minimum value for the cost function is RHC (0.0003 seconds), and then SA (0.0119 seconds). But the value they found are not small enough and can only be considered as some "local minimum", which are 0.43 and 0.34. On the other hand, both GA and MIMIC gain better results, and the best result is from MIMIC (the result for $[x, y]$ is [11,40]), which is the closest to 0. According to the performance comparison, even MIMIC is much slower than other three algorithms, for this optimization problem we still want to use MIMIC because of its lowest value.

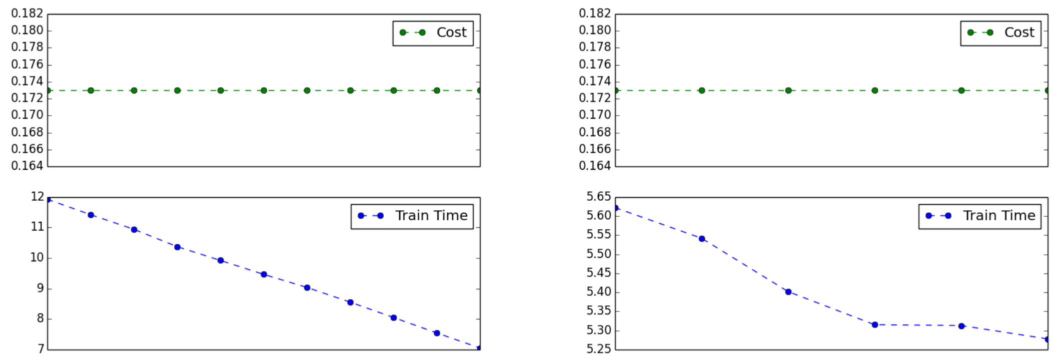
4.3 Improving MIMIC Performance

For this problem, we try to change two parameters to validate the performance of MIMIC. The two parameters are the samples size (samples to generate from the distribution at each iteration) and the percentile (the distribution to keep after each iteration). Figure 6 depicts the differences.

Table 3: Optimization Problem 2: Result Comparison After Optimization.

Algorithm	**Iteration**	**Total Cost**	**Training Time (s)**	**Solution**
RHC	20	0.43484	0.00033	[-33, -37]
SA	20	0.34893	0.01191	[6, 18]
GA	20	0.19260	0.25807	[-38, -40]
MIMIC	20	0.17304	106.95210	[11, 40]

Figure 6: Optimization Problem 2 with MIMIC: Performance Changes when Decreasing Sample-size (left) and Percentile (right).



We decrease sample number from 12,000 to 2000 with interval 1000, and decrease percentile from 0.9 to 0.7 with interval 0.05.

In our original model, we utilized `sample-size` of 10,000 and `percentile` of 0.9. As we can see from figure 6 and table 4, decreasing both parameters will decrease training time but has no obvious effects on the cost result. At the same time, the changes of sample size has much larger impact on training time of MIMIC than that of percentile. Therefore, according to the training time, a improved new MIMIC model is to use parameters that `sample-size` = 4,000 and `percentile` = 0.7.

5 Optimization Problem 3: Minimizing function with Single Optima

5.1 Problem Statement

In the last optimization problem, we aim to choose two integers x_1 and x_2 between $[40, 200]$ to find the only one local minimum value of a cost function:

$$f(x_1, x_2) = -exp(-1) + (x_1 - 100) \times (x_1 - 102) - exp(-1) + (x_2 - 100) \times (x_2 - 102). \tag{4}$$

The surface of this cost function is visualized in figure 7. For this problem 3, its minimum cost function value is at [101,101].

This optimization problem is interesting because it has a single optima point with a smooth surface. Although it is an easy problem, but we use this in order to highlight the advantages of RHC and SA: they indeed use the **local changes of the surface** (an information similar to the gradient of the surface) to guide the search. This characteristic of RHC and SA is particularly useful for cost functions with smooth surface. Further, this cost function only has a single optima, which avoids the local optima issues of RHC and SA. Therefore, this is an interesting problem to highlight RHC and SA.

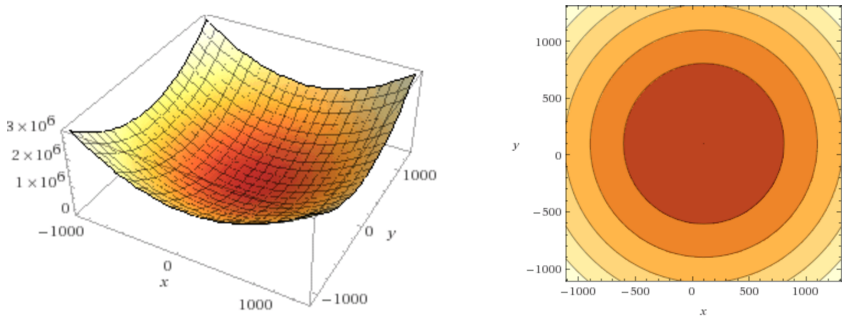
The domain is:

$$domain3 = [(40, 500)] \times 2. \tag{5}$$

Table 4: Optimization Problem 2 with MIMIC: Result Comparison on MIMIC with Different Parameters.

Param (sample-size & percentile)	**10,000&0.9**	**10,000&0.7**	**4,000&0.9**	**4,000&0.7**
Cost	0.17304	0.17304	0.17304	0.17304
Time (seconds)	10.3625	9.5863	3.9687	3.8648
Result	[11, 40]	[11, 40]	[11, 40]	[11, 40]

Figure 7: Optimization Problem 3: Cost Function.



5.2 Results from Optimization Algorithms:

Table 5 summarizes the results from RHC, SA, GA, and MIMIC algorithms on operation problem 3. We use 10 times iterations at this example case as well.

As shown in table 5, all algorithms reach good results. RHC, SA and MIMIC gain the exactly optimal solution which is [101, 101], while GA gets the solution [101,102]. As usual, the fastest way to find the minimum value for this cost function is RHC and SA. On the other hand, GA and MIMIC are about 4 times and 50 times slower than RHC and SA respectively. According to the performance and running time, RHC or SA is the best algorithm for this operation example. For the purpose of investigating more parameters in affecting performance rather than only "restarting times", we choose SA to perform parameter sweep for improving the result.

5.3 Improving SA Performance

Our original SA model use parameters `temperature`=100, `cooling-level`=0.99 and `learning-step`=1.

We first decrease the `temperature` from 1000 to 1 ([1000, 700, 400, 100, 70, 40, 10, 7, 4, 1]) to check the result changes. Then we increase the `cooling-level` from 0.1 to 0.99 ([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99]) to see the differences; lastly, we decrease the `learning-step` from 12 to 1 with the interval 1.

As results shown in figure 8:

(1) Decreasing `temperature` does not really affect the minimum cost, but it help to save running

Figure 8: Optimization Problem 3 with SA: Performance when Changing Temperature, Cool-level and Learn-step.

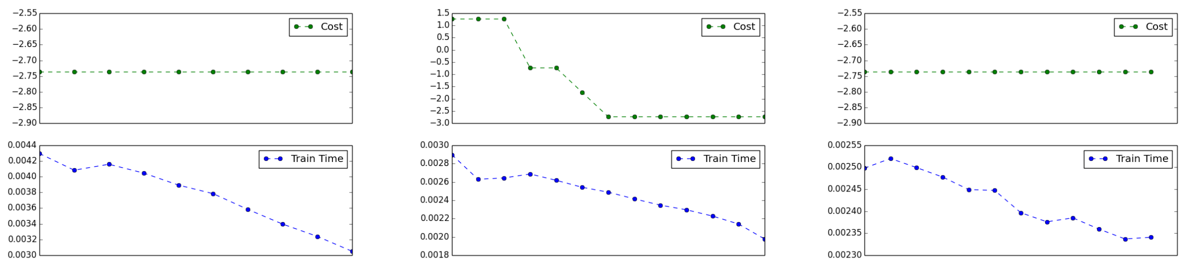


Table 5: Optimization Problem 3: Result Comparison.

Algorithms	**RHC**	**SA**	**GA**	**MIMIC**
Cost	-2.7357	-2.7357	-1.7357	-2.7357
Time	0.0072	0.0173	0.0313	1.3807
Result	[101,101]	[101,101]	[101,102]	[101,101]

Table 6: Optimization Problem 3 with SA: Improved Result Comparison.

Methods	**Original SA**	**Imporved SA**
Cost	-2.7357	-2.7357
Time	0.0325	0.0233
Result	[101,101]	[101,101]

time for SA model in the sense that lower temperature makes SA converge faster.

(2) In terms of increasing `cooling-level` of SA, the cost decreases as cooling level goes up. At the same time, the training time also goes down. It means that higher `cooling-level`, especially close to 1, is better for SA to converge to minima.

(3) For decreasing `learning-step`, though there shows no changes for the minimum value, the training time becomes less when decreasing the `learning-step`. It means that lower step gives shorter training time.

According to our discussion above, our original SA model has a relatively good parameter configuration. In addition, we re-configure the parameter value for `temperature`=10 as an improvement and re-run the model. The comparison between original SA model and improved SA models is given in table 6.

6 Summary

In this report, we investigated into the randomized algorithms RHC, SA, GA, and MIMIC. We first use RHC, SA, GA on training a neural network on Iris dataset, and compare the results with back-propagation algorithm. Further, we selected three optimization problems to validate the performances of RHC, SA, GA, and MIMIC. Furthermore, parameter sweeps are performed to improve the performance of selected algorithm for each problem. The results demonstrate that, while all four randomized optimization algorithms give very good results, the best algorithm for a specific problem is intrinsically determined by the problem itself.

References

[1] Toby, S., "Programming Collective Intelligence", 2007.

[2] Charles L. Isbell, Jr., "Randomized Local Search as Successive Estimation of Probability Densities", 1997.

[3] Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., "Machine learning: An artificial intelligence approach", *Springer Science & Business Media*, 2013.