Randomized Optimization

Ting-Yu Kang 903443289 tkang49

Abstract

In this homework, I am going to optimize the same Neural Network from the last homework. However, instead of using the traditional gradient descent method with back propagation, I will use Randomized Hill Climbing, Simulated Annealing, and Genetic Algorithm to update the weights of the Network. For the training result, I will observe the testing accuracies and try to explain them.

In the second part, I designed three interesting optimization problems and solved them by the above three algorithms and the fourth one: Mimic algorithm. From the experiments, I had found a lot of interesting point worth knowing. Not just learning from the theories in class, I prefer to get a deep understanding of a topic by actually manipulating the data.

Part 1: Neural Network Optimization

1. Dataset:
   (1) Description: I chose the **Spambase** dataset from the last homework. This dataset is collected from the real-world emails marked as "spam" or "non-spam" by individuals. The purpose is to identify an email is Spam or not from a set of attributes. (https://www.openml.org/d/44)
   (2) Dataset Overview:
      (a) Instances: 920
      (b) Attributes: 57
      (c) Class labels: 2 (1: spam, 0: non-spam)

2. Tools:
   (1) In this assignment, I used **mlrose** Python package to implement the underlying randomized optimization algorithms. I also imported **Scikit-learn** machine learning library and data structures from **Pandas** and **NumPy**.
   (2) The source codes can be executed both on **MacOS** environment and on **Google Colaboratory**.

3. Experiment Overview and Methodology
   (1) The dataset is binary classification problem, I separated it into **80%** training set and **20%** testing set.
   (2) The number of instances is 4601, to be more efficient without losing generality, I randomly sampled **20%** of data (920 instances) for this project.
   (3) The network structure is **(10, 10)**, which is chosen from homework 1 for it achieves the best prediction accuracy (94.39%) in 10-folds cross-validation among all candidate networks. The total number of weights in the network is $57 \times 10 + 10 \times 10 = \mathbf{670}$.

(4) Before training, I use MinMax scaler to preprocess the training data to make prediction more accurate. It might because the effects of noise and outliers are decreased.

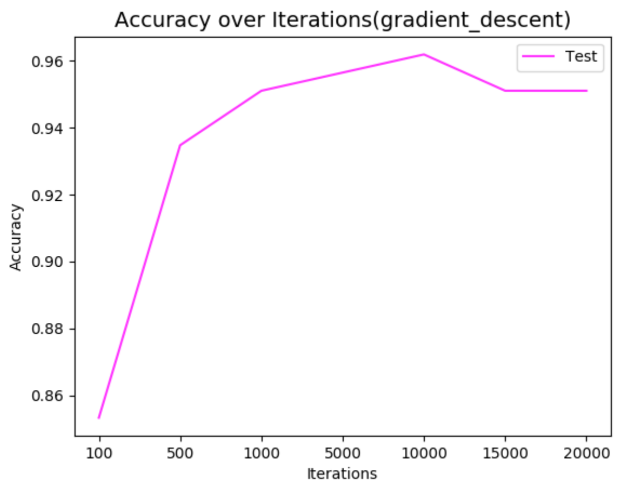
(5) For each algorithm, I **restart** 10 times to prevent the model being trapped in local maximum. after that, I choose the result with the highest accuracy as my result. This approach indeed help get more smooth accuracy line and describe the result of algorithm better.

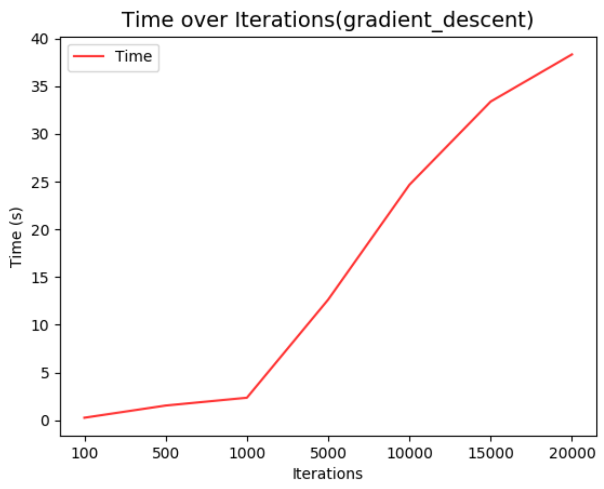
(6) In the following graphs, I will demonstrate the accuracy and execution time over iterations. The pink line is **testing** accuracy (the highest of 10 random restarts); and the red line is **execution time** (the average of 10 random restarts).

4. Gradient Descent (Baseline)

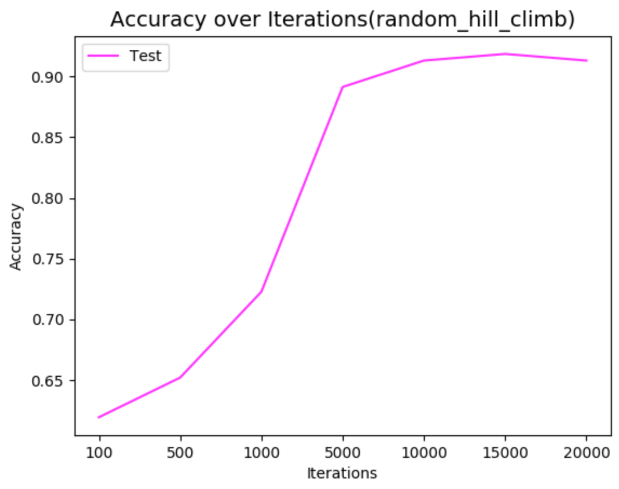
(1) To make a better comparison, I implemented the normal gradient descent neural network again as the baseline with the weights being updated by back propagation. The learning rate is 0.0001

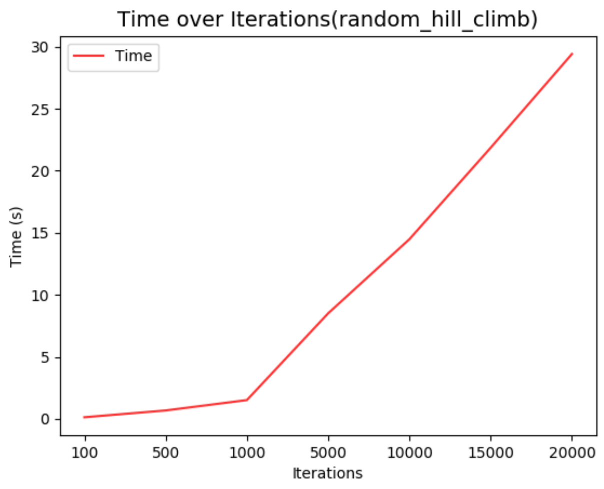
(2) We can see that the accuracy becomes steady from 1000 iteration. After that, there are only 2% growth from 1000 to 10000 iteration. After 10000 iterations, the accuracy drops, showing that the model might overfit the data.





5. Randomized Hill Climbing (RHC)





(1) We can see that the accuracy converges from 5000 iterations, larger than the gradient descent approach. After that, there are only 2% difference from 5000 to 20000 iterations.

(2) RHC is faster than gradient descent, because RHC only needs little time to compare between values. On the other hand, gradient descent needs more complex calculations to update weights through back propagation.

6. Simulated Annealing (SA)

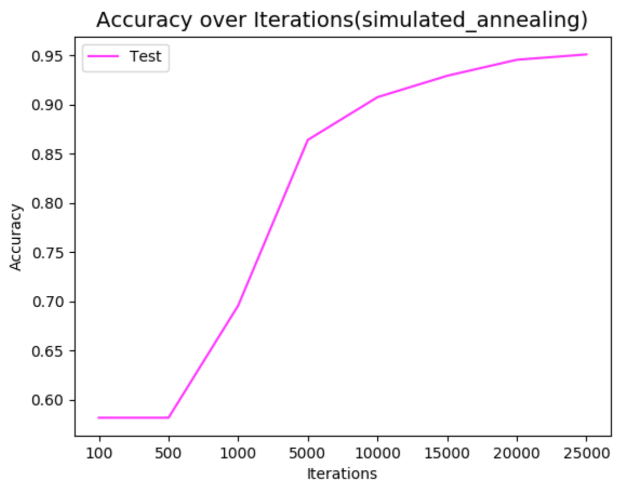
(1) I used the geometrically decay function to gradually lower the temperature parameter:

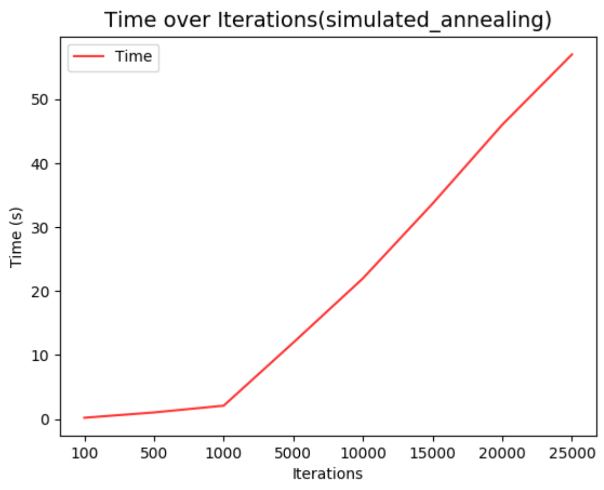
$$T(t) = max(T_0 \times r^t, T_{min})$$

Where $T_0(initial\ temp) = 1, r(geometric\ decay) = 0.99, T\_min(min\ temp) = 0.001.$

(2) We can see that the accuracy converges from 20000 iteration, more than the previous two approaches. It might because the temperature factor gives it more randomness to step in the bad states. Therefore, it needs more iterations to converge.

(3) The accuracy of SA is roughly as well as gradient descent, but the time for training and the number of iterations for converging is larger.



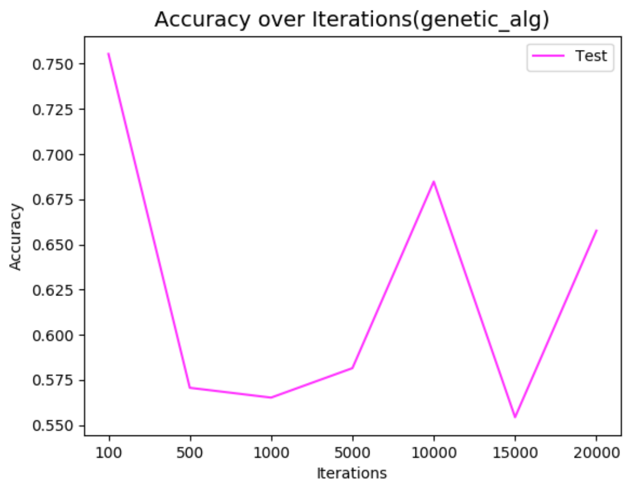


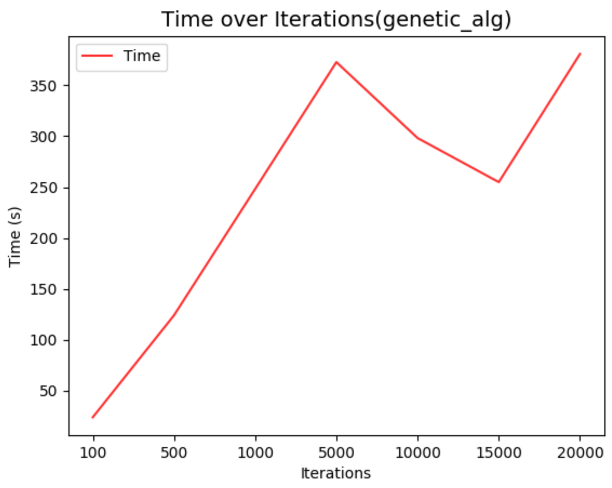
7. Genetic Algorithm (GA)

(1) I used $population\ size = 200$ and $mutation\ probability = 0.1$ to perform weight training.

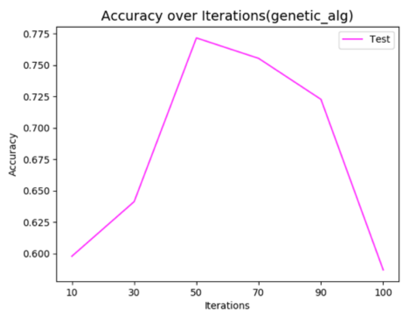
(2) We can see that the accuracy doesn't converge even at 20000 iteration. It is possible that it still needs more iterations to converge. However, it had already taken **2 hours** (20 hours for 10 restarts) to training and still fluctuate around 55% to 67%, while gradient descent only takes **1.55 seconds** to get 93%. Therefore, it is worthless to keep training.

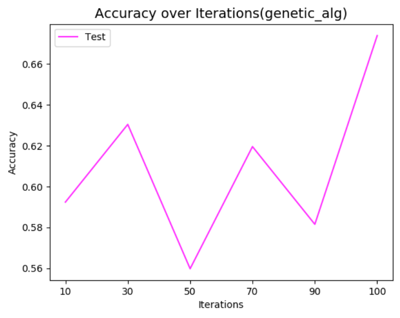
(3) GA is a time-consuming algorithm. According to the graph, GA calculates **30** iterations per second while gradient descent calculates **450** iterations per second.

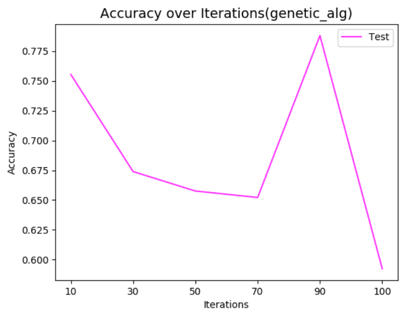




(4) In the above graph, the result fluctuates a lot. However, it seems that it reaches 76% at the first 100 iterations. Therefore, I run GA algorithm again and focus on the first 10 - 100 iterations:





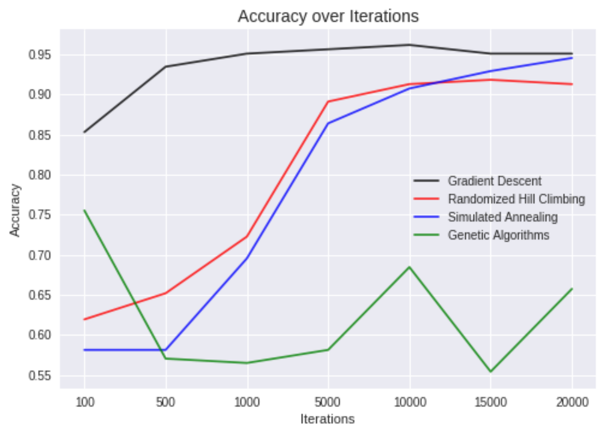


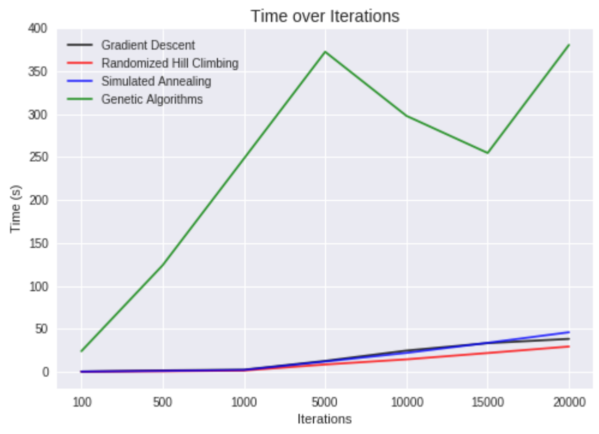
(5) From the above results, there are still no regular patterns to conclude. I can only say that the best accuracy GA can achieve is about 76%.

(6) The reason why GA performs bad might because it is not suitable for the dataset I used for this homework. Because spam mails have very unique characteristics, the values of attributes are concentrated and obvious. Therefore, doing mutation and crossover may is prone to jump to the worse state, making the result unpredictable and not that accurate. If I chose a more complicated dataset which is hard to be classified by normal learning algorithms, then GA's 76% accuracy might be excellent.

8. Summary

(1) The following graph combines the results of the above algorithms:





(2) First, the gradient descent (black) achieves the best testing accuracy (95.6%). Also, it only needs 500 iterations to reach 93% accuracy. The outstanding result shows that back propagation is the best choice for updating weights.

(3) RHC (red) and SA (blue) don't predict well in the beginning, but eventually get 90% accuracy at around 5000 iterations. However, their accuracies still lower than gradient descent. What's worse, the lines of RHC and SA are chosen from the best one of 10 random restarts because their accuracy fluctuates a lot. In fact, doing 10 random restarts is equal to repeat the same algorithm 10 times and take 10 times longer to calculate. Therefore, they are not that good for updating weights for neural networks.

(4) GA (green) performs the worst and takes the largest amount of time for model training. Therefore, I argue that GA is not suitable for the dataset I used this time, at least not efficient.

(5) Noted that randomized optimization algorithms may not perform so well on the relatively easy classification problems. Moreover, there are already simple and well-performed algorithms like gradient descent out there. Accordingly, next time when I use neural network, I will choose gradient descent instead of randomized algorithms.

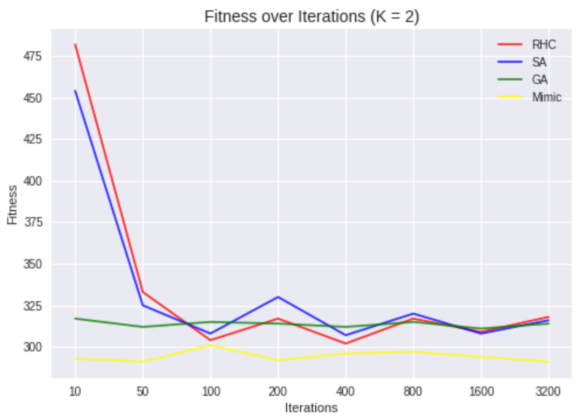
(6) The table below summarizes the four algorithms:

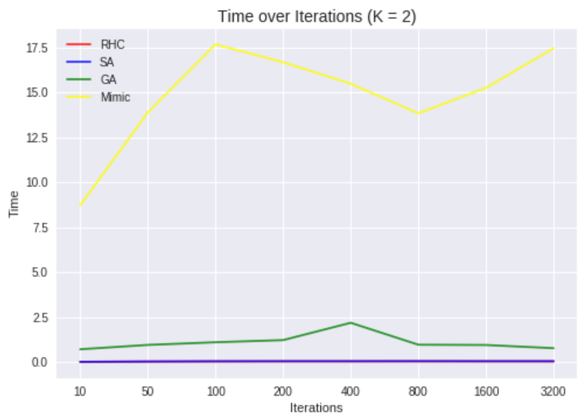
Algorithm	Best Testing Accuracy	Iterations and Time at Best Accuracy	Average iterations per second	Overall Ranking
Gradient Descent	95.6%	5000 (12.6s)	455.8/sec	1 (best)
Randomized Hill Climbing	91.8%	15000 (21.86s)	673.4/sec	3
Simulated Annealing	94.6%	20000 (46.03s)	440.9/sec	2
Genetic Algorithm	75.6%	100 (23.9s)	30.3/sec	4

Part 2: Optimization Problems

1. Experiment Overview and Methodology
   (1) Three problems I chose: K color optimization, Maximum mod function, and N-queens. In the following experiment, they will demonstrate the advantages of Mimic, Genetic Algorithm (GA), and Simulated Annealing (SA) separately.
   (2) I had carefully designed the problems for each algorithm. It is very interesting to map what I learned into the experiments on the real-world data.
   (3) To get more stable result of Randomized Hill Climbing (RHC) and SA algorithms, both of them were restarted 10 times to be more generalized and prevent stucking at local optimums.
   (4) In the following graphs, I will illustrate the fitness values and execution time over iterations for each problem.

2. **Problem 1**: K color optimization (Mimic)
   In this graph-based problem, Mimic is very powerful.
   (1) Problem definition:
   Given an undirected unweighted graph with the number of nodes and edges, the goal of this problem is to assign K different colors to each vertex to **minimize** the number of edges with the same color of adjacent nodes.
   (2) Input: 50 nodes with randomly generated edges in the form of an adjacency list.
   Output: The minimum number of illegal edges (with the same color of vertices)
   (3) For K = 2, there are only two colors (color 0 and color 1). It is equal to a bit string problem:





   (4) We can see that, Mimic (yellow) performs the best, and it converges in the first 10 iterations. In contrast, the other three algorithms cannot achieve better results than Mimic's.
   (5) The reason why Mimic performs so well is because it is good at solving problem with **special structures** and **dependencies** between nodes. From the class, Mimic uses dependency trees to approximate complex probability distributions and generate samples for the next iterations. Because this problem is graph-based, it is especially suitable for Mimic algorithms. The outstanding experiment result illustrates it.

(6) However, although Mimic only needs several iterations to converge, there is no free lunch. Mimic takes a lot more time per iteration than the others. Because for each iteration, it has to calculate the probability distribution, conditional probability, MST algorithms, and generate samples out of dependency trees. The heavy calculation makes Mimic algorithms time-consuming.

(7) Finally, there are fitness values of the algorithms with different K:

K	RHC	SA	GA	Mimic
2	309	307	311	291
3	196	180	203	178
4	137	137	151	121

It shows that Mimic outperforms on all Ks.

3. **Problem 2**: Maximum mod function (Genetic Algorithm)

I designed this mod-based problem to demonstrate the advantages of GA.

(1) Problem definition:

Given a bit string $\boldsymbol{S}$ containing only 1 and 0, find the **maximum** value after following calculation:

(a) Slice the bit string into 8-bit substrings and convert them into integers $\boldsymbol{n_i}$

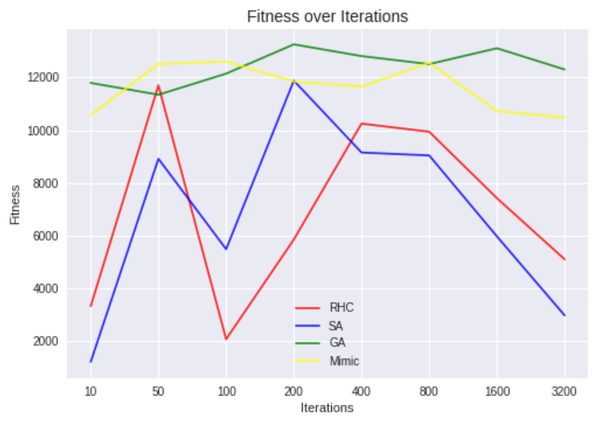
(b) Do mod and trigonometric calculations on each integer and multiply them:

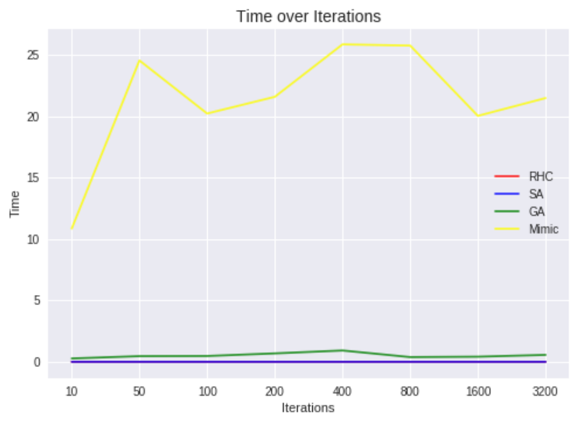
$$fitness(S) = \prod_i (n_i \; mod \; 13)^2 \; mod \; 7 + |\sin(n_i)|$$

(c) Noted that the sin term is to add some randomness to the fitness function.

(2) Input: 48-bit binary string

Output: The maximum fitness value after mod operation.





(3) From the graph, RHC and SA performs bad and are still unstable even at 3200 iterations. The reason for their big failures is because they always choose neighbors to determine their next steps, but this function is specially designed to have a lot of **local optimums** by the nature of mod operation. Therefore, RHC and SA are easy to get trapped.

(4) Genetic Algorithm is very stable, stunningly fast, and performs well just in the beginning. What's more, it even performs **better than Mimic** with less than 10% execution time.

(5) Unlike coloring problems with graph structures, it is not obvious to find a neighbor relationship in this case. Specifically, the fitness value will **change drastically** by just flip a bit. RHC and SA depends on the comparison of neighbors, so they will not be stable nor perform well.

(6) In contrast, GA and Mimic are able to mutate the data and come up with whole new generation through data crossover. Accordingly, they will get more opportunities to "**jump**" to the optimal fitness value. The result shows that GA is well-suitable for **discontinuous functions** like mod problems.

4. **Problem 3**: N-queens problem (Simulated Annealing)

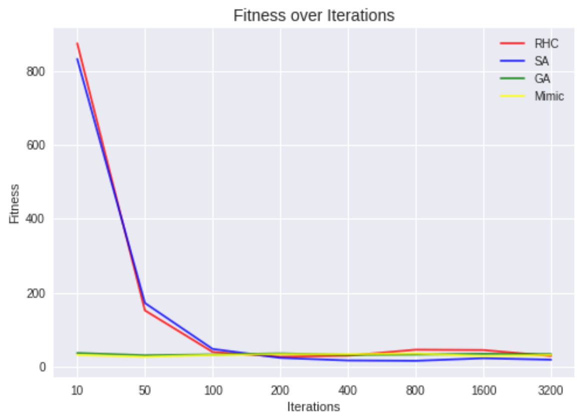
I use this simple but popular problem to show the advantageous of lightweight algorithms like SA.

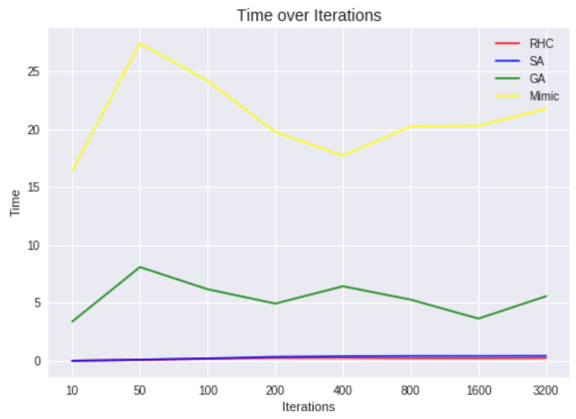
(1) Problem definition:

Given an $n \times n$ chessboard, find the position of n chess queens where they are not attacking each other, which means no two queens are in a same row, same column, or same diagonal).

(2) Input: 50×50 chessboard

Output: The **minimal** number of illegal positions where there are two queens attacking each other.





(3) From the graph, we can see that all four algorithm performs well and converges at around the first 100 iterations. After 200 iterations, SA performs the best. Although GA and Mimic also do well, they need 5 - 25 times longer than SA to approach the minimum.

(4) N-queens problem is a relatively easy problem compared to the previous two question. It has obvious neighbor relationship with a lot of **global optimums**, so RHC and SA can easily converge to one of the optimal solutions.

(5) For such easy problems with simple structure, we can choose lightweight algorithms such as SA or RHC. They can find as good answer as GA and Mimic do without requiring heavy calculations.

5. Summary

Following are the comparison of the underlying four randomized optimization algorithm:

Algorithm	Speed	Iteration to converge	Suitable for	Overall Ranking
Randomized Hill Climbing	Fast	hundreds	Simple, continuous, few local optimums	4
Simulated Annealing	Fast	hundreds	Simple, continuous, few local optimums	3
Genetic Algorithm	Medium	dozens	Graph-based, structural Problems with complex dependency of each attributes.	2
Mimic	Slow	dozens	Discontinuous functions with many local optimums.	1 (best)

Each algorithm has different characteristics and advantages. The choice of algorithms highly depends on the problems. For the complicated problems with strict requirement of optimal solutions, GA or Mimic is a preferable choice. However, for the problems where computation is expensive, RHC and SA could be better.

Reference

[1] Genevieve Haye. mlrose: Machine Learning, Randomized Optimization and Search. Retrieved from https://mlrose.readthedocs.io/

[2] Pushkar Kolhe. ABAGAIL. Retrieved from https://github.com/pushkar/ABAGAIL

[3] Ramdomized Optimization. Charles Isbell, Michael Littman. Retrieved from https://classroom.udacity.com/courses/ud262/lessons/521298714/concepts/5344086090923

[4] Rachit Belwariar. Test Case Generation I Set 4 (Random directed / undirected weighted and unweighted Graphs). Retrieved from https://www.geeksforgeeks.org/test-case-generation-set-4-random-directed-undirected-weighted-and-unweighted-graphs/