CS 7641 Machine Learning HW2

Hao-Jen Wang

Abstract

Many randomized optimization methods, as Randomized Hill Climbing (RHC), Simulated Annealing (SA), Genetic Algorithm (GA), and MIMIC, are developed to solve various classical NP-hard problems, and it is worth to find which method outperforms the others. In this project, I particularly interested in N-queen, TSP, and the Knapsack problem. They are so commonly seen that they become classical lecture contents in Algorithm courses and top interview questions according to Leetcode. Knowing how to address these problems is therefore practical to improve the coding ability.

1. Introduction

In this project, I will analyze the 32-queen problem deeply first and show how the problem size affects the model performance. To prevent explaining duplicated reasonings, the rest two problem, TSP and Knapsack, will be described briefly unless they are unique findings that is not mentioned in 32-queen. Then, I will show how these 4 methods perform on optimizing the weights of a neural network, compared to the gradient descent method. Lastly, I will deduct some facts according to the experiments.

2. Data Processing

The data used for TSP is FRI-26 composed of 26 cities and a minimal tour length 937. The data used for Knapsack is P-07 including a set of 15 weights and profits with a capacity 750 and an optimal profit of 1458. Both datasets are extracted from [3].

For better intuitions, the fitness scores in the three problems have a maximal value 0, and the optimization algorithms will try to maximize it. However, the fitness functions for N-queen and TSP are generally designed to be minimized. So, I multiply a minus sign to the original functions and plus the original global minimal values, and thus the problems are converted. For Knapsack, subtracting the optimal value to the original can also satisfy the setting of maximization problem with a maximal fitness score 0.

3. Problem Analysis

Before running the optimizations, analyzing the problem structures is necessary. For instance, N-queen becomes more complex as N is larger. From a related work [2], we know exact amounts of optimal solutions for different N. In Figure 1, it shows that more N yields lower probability of finding an arbitrary optimal solution at random (left), and makes the fitness scores farther away from the maximum (right). Notice the maximal fitness is 0 as described, and the distribution is generated from one million random states.

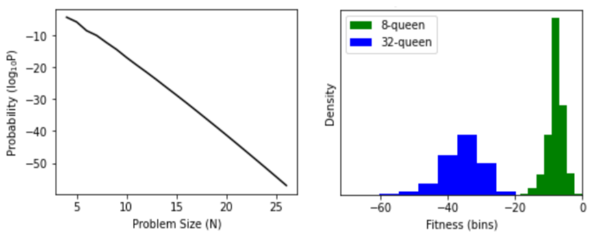

Figure 1. Attributes for N-queen

With the same trick, I can also analyze the state distributions for TSP and Knapsack. In Figure 2, the left figure is the distribution for

TSP, and the right one for Knapsack. The scores are also set with a maximal 0. Like N-queen, the distribution of TSP is also far from 0 except that the range of fitness is larger. Intuitively, the two problems may be unsuitable to be solved by the population-based algorithms such as MIMIC or GA, because the initial population with random individuals has low chance to produce offspring that suddenly approaches the optimum, and the vicious cycle continues. Increasing the size may solve the issue, yet the running time is huge.

As for Knapsack, the majority has the worst fitness, so it may be difficult for the incremental approaches like RHC and SA. It may be hard to escape from the majority states, yet population-based methods can generate offspring that does not belong to the majority states and therefore become easier to escape the trap. Given these observations, I guess that RHC and SA are more suitable for N-queen and TSP, whereas GA and MIMIC are better for Knapsack.

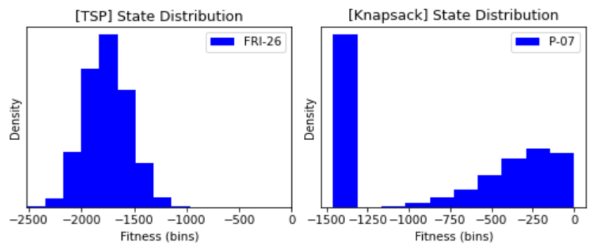

Figure 2. Attributes for TSP and Knapsack

4. Experiments

Given the intuitions from above, we shall now see the experiments. Most implantations are derived from *mlrose* package. It provides a parameter representing maximal attempts while trapping into a local optimum. The default value is too small, so I set it to be 100 throughout the experiments and will modify it if necessary.

A. 32-queen – randomized hill climbing

As mentioned earlier, the fitness score is set to be at most 0. The number of restarts is set to be 5. Though increasing it will guarantee to a better fitness score, my computational resource is limited, so I will freeze it. Plots of the fitness score versus iterations and processing times are shown in Figure 3.

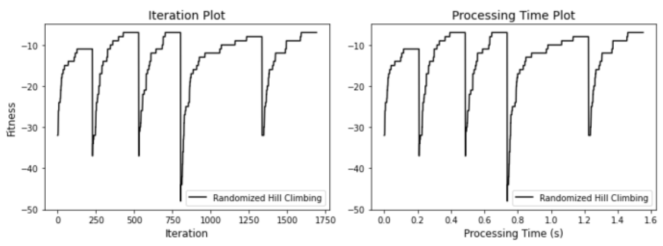

Figure 3. [32-queen] RHC performance

In 32-queen RHC, the best fitness is -7.0 with a corresponded running time 1.55s. Note that RHC is the most naïve method among all optimization methods since it restarts without any prior knowledge. Therefore, there is no surprising that other methods will outperform it.

B. 32-queen – simulated annealing

Now comes to the part of SA. The result is shown in Figure 4.

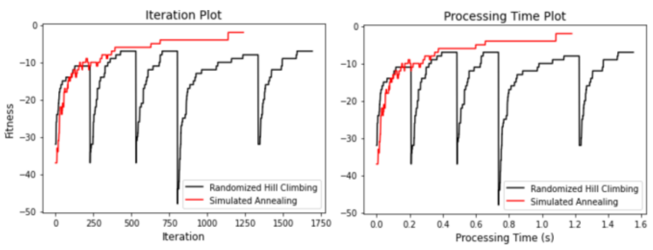

Figure 4. [32-queen] SA performance

We can see SA method outperforms RHC in both fitness score and processing time. It is due to that RHC restarts at random totally, whereas SA has been embedded with the knowledge of approaching to a better solution with a proper temperature. The optimal SA yields -2.0 fitness score with a corresponded running time 1.18s.

Yet, SA still has many hyper-parameters to be tuned. I focus on the initial temperatures and different decay functions here. In theory, larger initial temperature produces more iterations and makes a model a better chance to approach the global optimum. Figure 5 shows how the initial temperature affects the scores, where the plot on the left gives a detailed look at how 3 different initial temperature affects the fitness curve, and the plot on the right is the overall best fitness as a function of logarithm of initial temperatures from 5 repeated experiments.

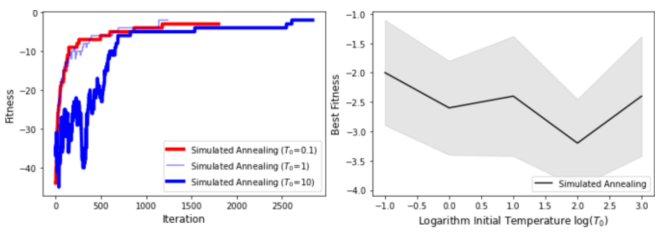

Figure 5. [32-queen] SA initial temperature

It turns out that higher initial temperature guarantees neither more iterations nor a better fitness score. The statistical result from the right graph shows the fitness score is independent to the initial temperature. The reason might be due to that 32-queens has too few optimal solutions. As shown in Figure 1, increasing the size will exponentially decreasing the chance of finding an arbitrary optimal solution at random. When the size is 32, it is estimated that the probability is lower than 10^{-64}! Thus, SA may take lots of time for, yet in the end the temperature is low enough for SA to explore further.

Other facts like temperature decay scheduler may also affect how to find the target. Figure 6 shows how different decay functions, including Exponential, Geometrical, and Arithmetic, influence the fitness scores.

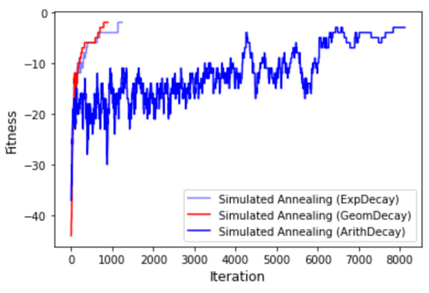

Figure 6. [32-queen] SA scheduler

Three schedulers yield the same scores -2 and fails to meet the optimum 0. To this end, the performance of SA is not affected by initial temperatures or decay function, yet it still has the best fitness score currently. Afterall, what we care is the relative performance compared to the 4 optimization methods. In this sense, I will not revise it unless some other models turn out to perform better than SA.

C. 32-queen – genetic algorithm

GA is a population-based algorithm. Each iteration for GA contains several evaluations. To better compare with other methods, here I focus on the number of iterations instead. With a default setting, the result is shown in Figure 7.

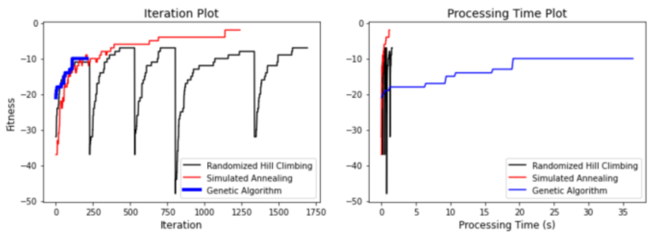

Figure 7. [32-queen] GA performance

Unfortunately, both the fitness and running time is the worst among all. It is reasonable that the computation is slow because each iteration contains a bunch of evaluations. However, the default parameter setting might not be optimal. Related parameters include the population size and the mutation probability. With other being fixed, first I show how the size affect the score with 5 repeated experiments in Figure 8.

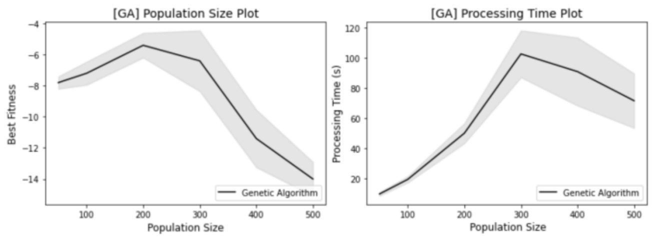

Figure 8. [32-queen] GA population size

We can see that the original setting (N=200) is still the optimal. While fewer population have less feature expressions, larger sizes in this case turn out to decay the fitness score! In theory it is unreasonable, so it is necessary to find out some other factors that also affects the performance. Another factor includes the mutation probability, and Figure 9 shows the result.

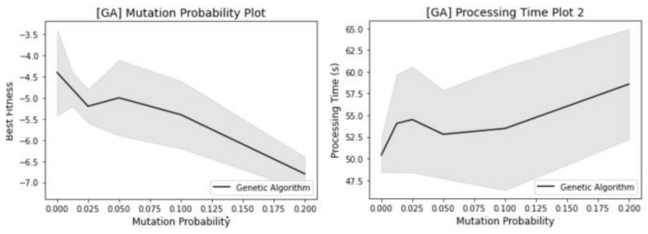

Figure 9. [32-queen] GA mutation probability

Obviously, **no mutation is the best**! Maybe mutating the individual incurs some noises, and as 32-queen has low probability of approaching to the global optimum, the offspring may yield worse fitness score. In this experiment, GA with no mutation produces -5.0 fitness score, which is lower than SA but finally higher than RHC.

With this GA (namely GA v2), we can again make comparison shown in Figure 10.

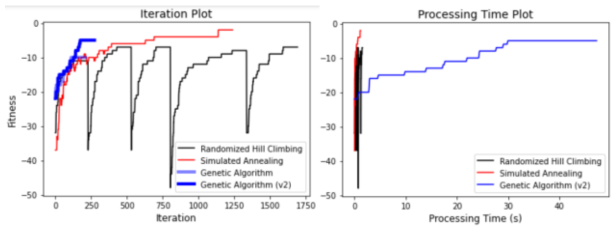

Figure 10. [32-queen] GA v2 performance

To this end, SA is still the best. RHC has a lower fitness score than GA v2, yet it is much faster. Therefore, GA is not a proper method for the 32-queen problem.

D. 32-queen – MIMIC

There are many parameters need to be tuned for MIMIC. For simplicity, the default setting is used at first. The comparison is in Figure 11.

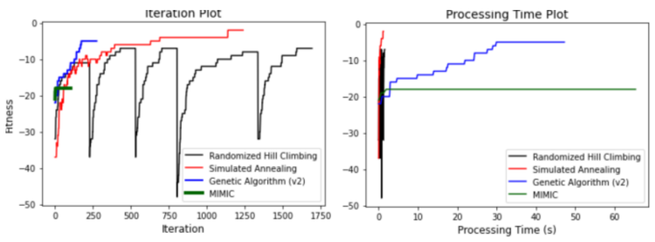

Figure 11. [32-queen] MIMIC performance

Sadly, MIMIC performs the worst yet very costly. Some tunable parameters like population size and probability of keeping samples shall be tuned. The effect of size is shown in Figure 12.

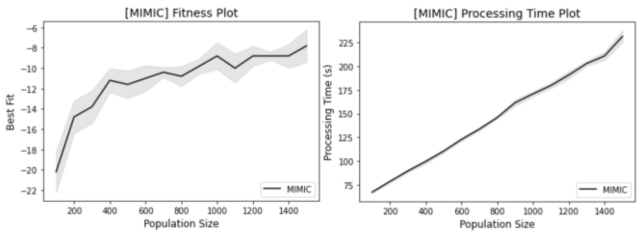

Figure 12. MIMIC population size

Again, the experiment is repeated 5 times for stability. Unlike GA, increasing the size here can guarantee a best fitness score. It means that more population can maintain the distribution without affected by the noise. In the left plot of Figure 12, it seems that adding more size may further increasing the score, yet due to a limited computational time, I restrain maximal size here to be 1500. I name it as MIMIC v2 with such a setting. Now compare again in Figure 13.

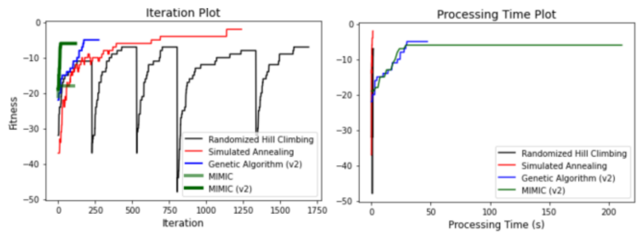

Figure 13. [32-queen] MIMIC v2 performance

The fitness score is improved from -18 to -6, yet it still underperforms SA and GA. Like GA, larger size may further boost the performance, yet it is impractical due to the running time. To sum up, we may compare all methods together in Table 1. It shows that SA is the best in both fitness score and processing time.

Table 1. [32-queen] Overall comparison

Factor / Method	**Fitness Score (maximum: 0)**	**Processing Time (s)**
Randomized Hill Climbing	-7.0	1.55
Simulated Annealing	**-2.0**	**1.18**
Genetic Algorithm	-5.0	47.39
MIMIC	-6.0 (*)	213.06

(*) Due to limitation, the result is not optimal

E. *N-queen Generalization*

We know SA is the best method in 32-queen, and it is worth to justify if SA is always the best in N-queen no matter the problem size. With all the same model setting in previous experiments, Figure 14 shows the fitness score and running time with respect to different problem size.

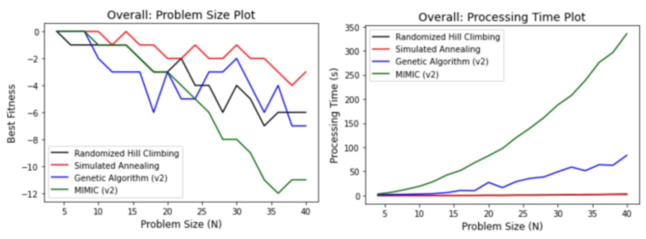

Figure 14. [N-queen] Model comparison

Whatever N is, SA outperforms all the other methods regarding both the fitness and runtime. RHC and GA ties in fitness, yet the runtime for GA is linear to the problem size. MIMIC is the worst among all, and maybe the parameter shall not be fixed across different N. The difficulty of tuning MIMIC is that the explore space is huge. As a result, MIMIC is impractical for complex problem space given limited resources.

F. *TSP 26 – randomized hill climbing*

As described previously, the global optimal value is converted to 0, and the fitness score is to be maximized. In this problem, I found that the peaks are not flat as shown in the gray line of Figure 15. Too few maximal attempts may end up restarting the exploration too early, and thus it is more likely to be trapped into a local maximum. In TSP 26, I increase the number of attempts from 100 to 1000. The result of RHC with different attempts is shown in Figure 15.

With maximal attempts set to 1000, it yields -157 fitness score with a running time 37.07s. I will fix such a number in the remaining TSP experiments for simplicity.

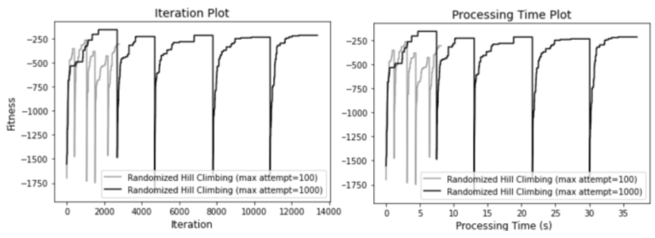

Figure 15. [TSP 26] RHC performance

G. *TSP 26 – simulated annealing*

With the default parameter setting and 1000 maximal attempts, the initial result of SA yields a fitness score -190, which is than RHC. In the ablation studies on how initial temperature or decay function affects the scores, the result in Figure 16 shows that it is unclear which setting is better than the others. The left graph shows how different initial temperature affect the score, and the right shows different decay schedulers. The light-blue line is the default setting.

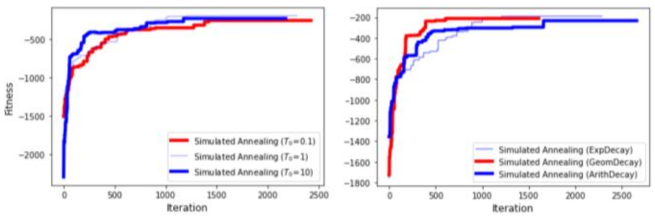

Figure 16. [TSP 26] SA different settings

After various trials, I found that decreasing the minimum temperature threshold can boost the fitness score. Smaller threshold value has the potential of escaping from local minimums. Figure 17 shows that decreasing the threshold can increase the overall fitness score.

For a better visualization, I discard the line segment from early iterations. Lower threshold can yield a better fitness score. With the setting of initial temperature 10^{-4}, I define it as SA v2 (the version ID is reset in different problems). It yields -101 fitness score and exceeds RHC in both the fitness score and running time.

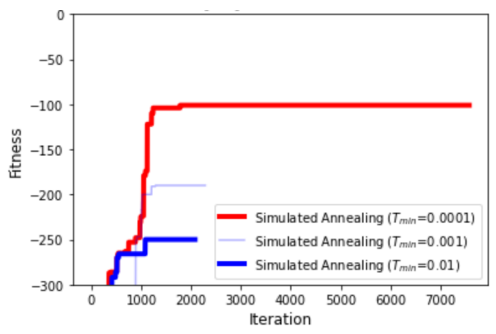

Figure 17. [TSP 26] SA minimal temperature

H. *TSP 26 – genetic algorithm*

Again, with a default parameter setting and a maximal attempt 1000, the result yields -316 fitness score with a runtime 1687s. Obviously, it shall be tuned properly.

First, I try to tune the mutation probability, but the performance becomes even worse (-353). Next, I try to tune the number of populations, and find out increasing it can boost the fitness score! The result is shown in Figure 18, where the light-blue GA curve is the setting with a size 200 and GA v2 is the setting with 400.

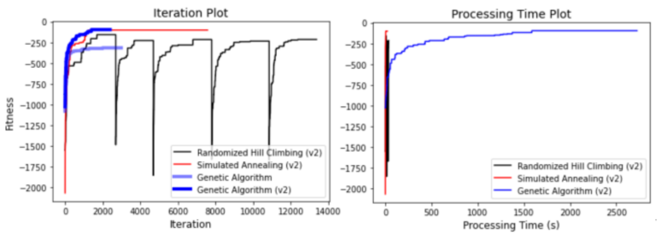

Figure 18. [TSP 26] GA population size

Amazingly, GA v2 yields a slightly better score (-96) than SA (-101). However, it requires 1.17 hours to run. Although increasing the size may further boost the performance, again it is very impractical in this problem. If time permits, I may do it in the future.

I. *TSP 26 – MIMIC*

MIMIC tends to be the slowest and hardest to tune among all the algorithms. At first, I use the default parameters, and it yields only -763 fitness score with a runtime 1324s. Based on the N-queen example, one effective way to improve the fitness score is to increase the population size. Figure 19 shows the relationship between the population size and the fitness score.

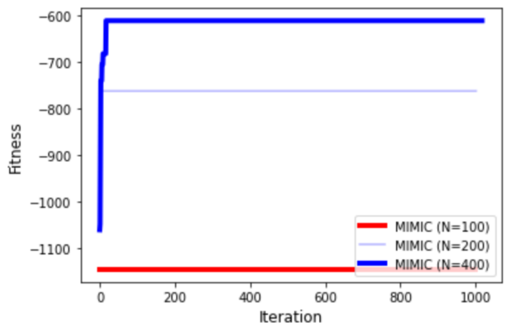

Figure 19. [TSP 26] MIMIC population size

Clearly, larger size can always make a better optimization task. With a limited budget, I tried my best at expanding the size from 200 to 1000. The overall comparison is shown in Figure 20, where MIMIC v2 represents the size of 1000.

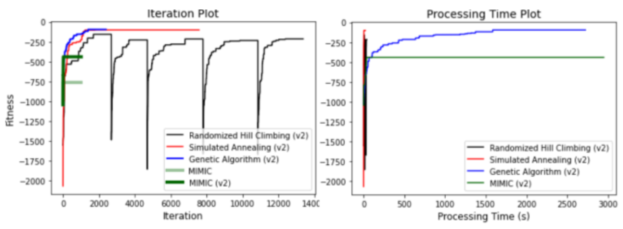

Figure 20. [TSP 26] MIMIC v2 performance

Now the MIMIC makes some improvement compared to the default setting of the size, from -763 to -440. However, it still underperforms the other algorithm. Actually, it might be better to decrease the maximal attempts here. In future work, I will try to further increase the size to see if the fitness can boost further. To sum up, the overall comparison is shown in Table 2.

Table 2. [TSP 26] Overall comparison

Factor / Method	**Fitness Score (maximum: 0)**	**Processing Time (s)**
Randomized Hill Climbing	-157	37.07
Simulated Annealing	-101	**<u>20.01</u>**
Genetic Algorithm	**<u>-96</u>** (*)	4229.91
MIMIC	-440 (*)	2953.13

(*) Due to limitation, the result is not optimal

J. *Knapsack 15*

In N-queen and TSP, it is unfortunate that MIMIC does not outperforms any others. As I described earlier, MIMIC is unsuitable for a too complex problem with a limited running budget. Therefore, I decide to choose a problem with a smaller candidate space here. Unlike previous problems, there is only a single optimum with 2^{15} possibilities, and in Section 2 I illustrate that this problem might be suitable for populational-based algorithms.

To avoid duplication, I just briefly introduce how to produce the result from the 4 algorithms and combine the performance plot into a single figure shown in Figure 21.

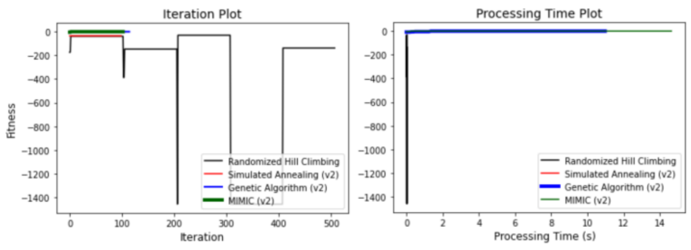

Figure 21. [Knapsack 15] All performances

In RHC, increasing the maximal attempt is useless since it has already reach the peak with fewer number. Thus, the default number 100 is used across all the algorithms. In SA, changing the scheduler from Exponential to Arithmetic can boost the fitness score, from -114 to -37. It is because Arithmetic scheduler decays slower and is better to escape from the majority states with the worst fitness (mentioned in Section 2). On the other hand, changing the initial or the minimal temperature do not yield a better score. To this end, RHC yields -30 fitness score with a running time 0.05s, whereas SA yields -37 score with a time 0.01s. Both methods are unable to reach the global optimum.

In both GA and MIMIC, however, adding the population size from the default 200 to 1000 can reach the global optimum! The comparison of these 4 methods, regarding the fitness score and the running time, is shown in Table 3.

Table 3. [Knapsack 15] Overall comparison

Factor / Method	**Fitness Score (maximum: 0)**	**Processing Time (s)**
Randomized Hill Climbing	-30	0.05
Simulated Annealing	-37	**0.01**
Genetic Algorithm	**0**	11.07
MIMIC	**0**	14.78

According to the results from N-queen, TSP and Knapsack, SA has the fastest running time. In N-queen, the best fitness also comes from SA. In TSP, the best fitness score comes from GA, and MIMIC shall have the best in theory, yet it is heavily restricted by the running time. In the Knapsack problem, both GA and MIMIC reach the global optimum. Therefore, there is no such an optimization method that is so-called the best.

K. *Neural Network*

Here we consider a continuous optimization problem: updating the weights of NN. I will use the Breast Cancer Wisconsin Dataset provided by *scikit-learn* [4]. The model will be built based on a 2-layer neural network architecture, where each layer contains 10 nodes. The features are scaled within [0,1], and the data are split into 80% for training and 20% for test. All hidden layers use ReLU as activation functions, and the loss function is based on SGD.

One way to have intuition on how to tune the parameters can be refer to the setting of NN in *scikit-learn*. By doing so, I can first refer the setting to build an equivalent SGD model from *mlrose*. To address the randomness issue, I will run the experiment 10 times and compute the mean and standard deviation of performances.

After tuning the *scikit-learn* version SGD, it has a 96.4% test accuracy in average with std 0.007. I use same parameters for NN in *mlrose*, yet the training curve turns out to rise too fast and does not improve, which means the learning rate is too high. With a smaller learning rate, it converges and yields 97.1% mean accuracy with std 0.007. The average runtime is 1.29s.

Next, we consider RHC and SA. They both require 10000 iterations and a step size 0.01 to be converged. RHC has 96.93% mean accuracy with std 0.01, and SA yields 97.02% average accuracy with std 0.01. To visualize the training curve, I draw 3 out of 10 sample results and plot in Figure 22, where the upper 3 figures are the result from RHC, and the lower 3 are SA. They have the same number of iterations and similar fitness score, yet RHC needs 78.89s to compute in average and SA requires only 20.21s.

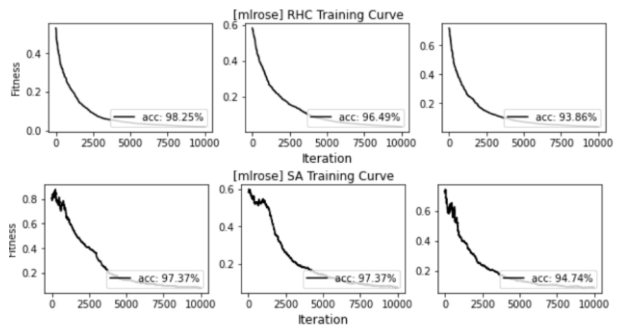

Figure 22. [NN] RHC and SA curves

For GA, the optimal population size is 400, and increasing it yields an identical fitness score. The average test fitness is 96.5% with std 10^{-16}, which is slightly lower than SA but much more stable. Figure 23 shows the training curve with 400 populations (upper) and 800 populations (lower). Despite different curve shapes, they have the same test accuracies.

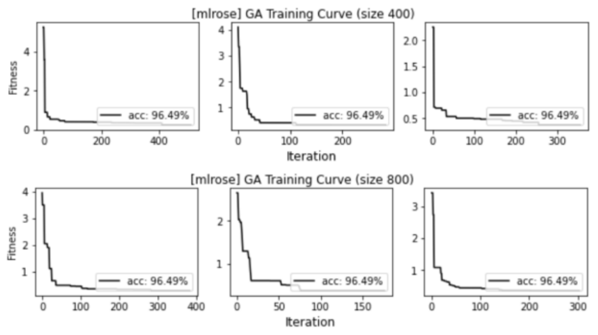

Figure 23. [NN] GA population size

Table 4 shows that overall comparison of all models to optimize the weight of NN. All the 4 models perform decently, and the SGD method still has the highest performance. Yet, GA can produce the most stable result.

Table 4. [NN] Overall comparison

Factor / Method	**Fitness Score (mean/std)**	**Processing Time (s)**
Randomized Hill Climbing	0.9693/0.0126	78.8909
Simulated Annealing	0.9702/0.0098	0.2814
Genetic Algorithm	0.9649/**0.0000**	54.4438
Gradient Descent	**0.9711**/0.0079	**0.0254**

Now we have the experimental result of NN. As for how the randomized optimization work for continuous problems, it needs to define the step size at first. For each iteration, one of the elements in the current state will add or subtract the number of the step size. Such a modification yields a new neighbor, so the problem can be viewed as discrete in this sense.

5. Discussion

Based on previous results, we may be able to draw some insights for each method:

Randomized Hill Climbing

- Few parameters are needed be tuned.
- The number of maximal attempts depends on how flat the peaks in the fitness curve are.
- Increasing the number of restarts can yield a better performance yet may not be effective.

Simulated Annealing

- It performs well across multiple problems.
- It has more iterations yet fastest to compute.
- It may be unsuitable for problems that have the majority states with the worst fitness. In such cases, Arithmetic decay scheduler may perform better than the Exponential.
- Decreasing the initial temperature or the minimal threshold do not guarantee a better model performance.

Genetic Algorithm

- It is the most stable method to optimize NN.

- Increasing the population size is effective as long as the fitness function is not sharp toward some local optimums.
- Mutating the expression is harmful when the fitness score distribution is far away from the global maximal solution.

MIMIC

- It has least iterations yet slowest to compute.
- The parameters should be tuned for different problem size in the same problem.
- Increasing the population size is effective across multiple problems.

6. Conclusion

This project tries to highlight advantages of four optimization algorithms across three kinds of NP-hard problems. The problem set includes N-queen, TSP, and Knapsack problem. Because academic institutions and industries often apply these problems to justify the coding ability, they are interesting and worth to investigate.

In theory, MIMIC shall yield the best result since it has the most degree of freedom (i.e., the number of parameters), it is costly and hard to tune when the problem complexity is high. Key findings of this project include how to choose a proper optimization method given a problem distribution. Incremental approaches like RHC and SA perform relatively better as the overall fitness distribution is far away from the global optimum. Populational-based method like GA and MIMIC perform relatively better when the states with the worst fitness are the majority.

Reference

[1] https://mlrose.readthedocs.io/en/stable/index.html

[2] http://www.durangobill.com/N_Queens.html

[3] https://people.sc.fsu.edu/~jburkardt/datasets/

[4] https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html