CS 7641 Assignment 2

Yijun Xie

***Abstract*—This report explores the performance of three random optimization algorithms—Random Hill Climbing, Simulated Annealing, and Genetic Algorithm—applied to discrete optimization problems and neural network weight optimization. The study involves implementing these algorithms and analyzing their behavior and effectiveness on two custom optimization problems and a neural network trained on the German Credit dataset from Assignment 1. Through this analysis, we aim to understand the strengths and weaknesses of each algorithm and identify the optimal conditions for their application.**

I. Introduction

The purpose of this project is to explore the behavior of random search algorithms under various circumstances. Understanding an algorithm requires not only reading about it or implementing it but also observing its performance across different scenarios. In this report, we implement three random optimization algorithms—Randomized Hill Climbing, Simulated Annealing, and Genetic Algorithm—and apply them to two custom optimization problems and a neural network optimization task. This study aims to highlight the strengths and weaknesses of each algorithm, providing a comprehensive analysis of their performance and suitability for different types of optimization problems. Our hypotheses are that:

1) The hyper-parameters will significantly affect the performance of these optimization algorithms.
2) There will not be a single optimization algorithm that outperforms all others.

II. Random Optimization Algorithms

In this report, we compare the performance of the following three random optimization algorithms.

A. Random Hill Climbing Algorithm

The random hill climbing algorithm starts with an initial random state and iteratively explores neighbors to find a better one. If a better neighbor is found, it becomes the new state solution. If no improvement is observed for a certain number of iterations, a random restart is triggered. This process continues until a maximum number of iterations is reached.

B. Simulated Annealing Algorithm

The simulated annealing algorithm also starts with an initial state and iteratively explores neighboring states. A neighbor solution is accepted if the score is improved, or with a certain probability if it isn't. The probability of accepting a worse solution decreases over time according to a cooling schedule. This helps the algorithm escape local maxima and find a global maximum.

1) Cooling schedule: In this report, we use an exponential decaying cooling schedule, i.e., the acceptance probability is defined as

$$P(x, x', T) = \begin{cases} 1 & \text{if } f(x') \geq f(x) \\ \exp\left(\frac{f(x')-f(x)}{T}\right) & \text{otherwise} \end{cases}$$

where x' is the new state.

C. Genetic Algorithm

The genetic algorithm is inspired by natural selection. It starts with a population of random states instead of just one random state like the previous two algorithms. The best K states are picked based on their scores, and the next generation of the population is generated through crossover and mutation. The process continues until a maximum number of iterations is reached, and the final solution is picked from the latest population.

1) Crossover: In this report, we implement the uniform crossover to combine the two parent states to generate a new offspring state. The logic works as follows:

1) For each element in the parent solutions, a random number between 0 and 1 is generated.
2) If the random number is less than 0.5, the element from parent 1 is chosen; otherwise, the element from parent 2 is chosen.

This process creates a child state that inherits elements from both parents, with each element having an equal probability of being selected from either parent.

2) Mutation: To implement the mutation logic, we:

1) Randomly select an index between 0 and the length of the state.
2) For the element at the selected index, add a small perturbation (a uniformly distributed random number between -0.1 and 0.1).

The mutation step introduces more variability and prevents the algorithm from getting stuck in local maxima, allowing it to explore the full search space.

III. Problems

In this report, we apply random optimization algorithms to two types of problems: discrete optimization and neural networks.

A. Discrete Optimization

We focus on two specific discrete optimization problems here: the N-Queens Problem and the Traveling Salesperson Problem (TSP).

1) N-Queens Problem: In the N-Queens problem, our objective is to place N queens on an N-by-N chessboard so that they cannot attack each other. Hence, we want to find a placement such that there will be no two queens in the same row, column, or diagonal. To simplify the problem, we assume N is fixed to be 8.

- **Random Neighbor:** Notice that the state for an N-Queens problem can be represented as a list of integers from 0 to N-1 in random order. To generate a random neighbor, we randomly swap two items in the list to ensure the random neighbor is still a valid state.
- **Score:** To evaluate the goodness-of-fit, we count how many overlaps occur in rows, columns, and diagonals. If it is an optimal state, the count should be zero. Since the algorithms are designed to maximize the score, we define the score of our N-Queens problem to be the negative of the overlapping counts.

2) Traveling Salesperson Problem: In the Traveling Salesperson problem, we want to find a route that will allow a salesperson to traverse a given list of cities exactly once and return to the original city. The goal is to minimize the traveling distance. In this report, we assume there are 8 cities on an 8-by-8 grid.

- **Random Neighbor:** Similar to the N-Queens problem, the state of a TSP is represented as a list of integers from 0 to N-1 in random order. Hence, a random neighbor is generated by swapping two items in the list to ensure the random neighbor is still a valid state.
- **Score:** The traveling distance is calculated as the sum of Euclidean distances between the neighboring cities in the state and between the last city and the first city. Since the algorithms are designed to maximize the score, we define the score to be the negative of the total traveling distance.

B. Neural Network Optimization

We also discuss a problem defined in the continuous space. Consider the German Credit problem we worked on in Assignment 1, where we are fitting a neural network with three layers (128, 64, 32 nodes on each layer). We also use a customized loss function where the cost of classifying an object with a `bad` label into `good` is 5 times as high as classifying an object with a `good` label into `bad`. Instead of implementing the commonly used gradient descent-based methods, we will use random optimization algorithms to find the optimal weights that will minimize the cost.

- **Random Neighbor:** For the neural network optimization problem, the state is the flattened weights of all nodes. We generate a random neighbor by adding a small perturbation (a uniformly distributed random number from -0.1 to 0.1) to a randomly selected node. In this way, we can preserve the structure of the neural network under training.
- **Score:** For German Credit data, we define a customized loss as the number of false positives plus 5 times the number of false negatives. We define the score as the negative of this loss so that the optimized state will minimize the loss.

IV. Experiment Setup

A. Optimization Problem Setup

For the three optimization problems we discussed in this report, we consider the following settings:

1) N-Queens Problem:

- The board size is 8-by-8.
- The score is the negative of total overlaps in rows, columns, and diagonals.

2) Traveling Salesperson Problem:

- The number of cities is 8.
- The grid size is 100-by-100.
- The traveling distance is measured by Euclidean distance.

3) Neural Network Optimization:

- The neural network has three layers, with 128, 64, and 32 nodes on each layer.
- 80% of data are used as the training set for finding the optimal weights, and 20
- The scoring function is a weighted misclassification loss, where false positives count 5 times more than false negatives.

B. Hyper-parameters for Optimization Algorithms

Here we present the hyper-parameter spaces for each of the three optimization algorithms.

1) Random Hill Climbing:

- Number of neighbors: 3, 5, 10
- Number of no improvements before restarting: 0, 10, 50

2) Simulated Annealing:

- Initial temperature: 1, 10, 100
- Decaying factor α: 0.99, 0.999
- The restart limit is fixed to be 100

3) Genetic Algorithm:

- Mutation rates: 0.1, 0.2
- Number of individuals kept in each iteration: 10, 20, 30
- The population size is fixed to be 200

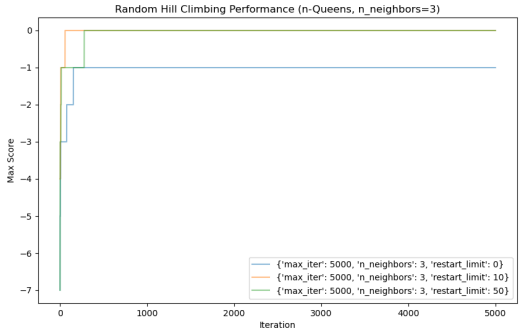
V. Results

A. N-Queens Problem

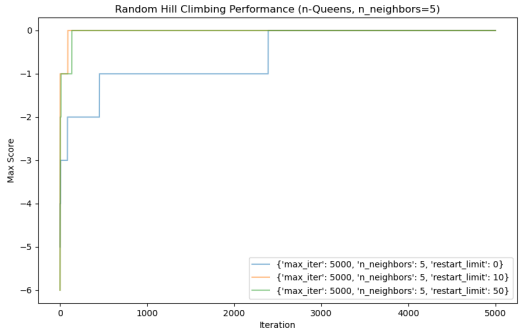
In Figure. 1, Figure. 2, and Figure. 3 we can find the results for the N-Queens problem. We observe that:

- Both the random hill climbing algorithm and simulated annealing algorithm can reach the optimal state with a relatively smaller number of iterations. However, the genetic algorithm does not perform as well as the other two algorithms. Out of six different combinations of hyper-parameters, only one reaches the optimal state.
- One possible explanation is that for the N-Queens problem, the state space is relatively constrained (here we require all elements to only appear once). However, the mutation process of the genetic algorithm does not necessarily guarantee this.

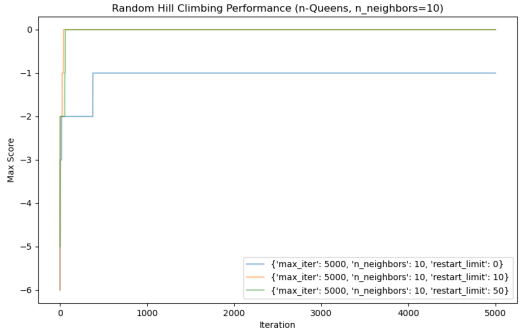
- Furthermore, the genetic algorithm generally makes larger moves compared to the other two algorithms which will only proceed one step. Hence, for a problem with a relatively small state space (in total $N!$ possible states), the genetic algorithm is more likely to miss the optimal solution.



(a) Random Hill Climbing - neighbor=3



(b) Random Hill Climbing - neighbor=5



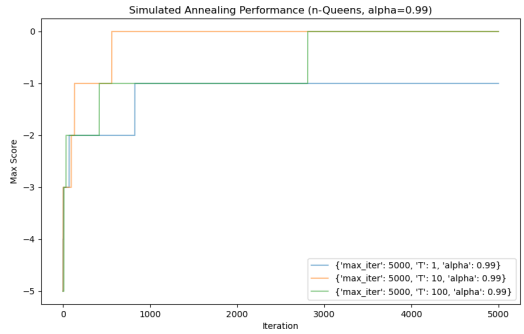
(c) Random Hill Climbing - neighbor=10

Fig. 1. The scores for 8-Queens problem from random hill climbing algorithm. An optimal solution should have score = 0.

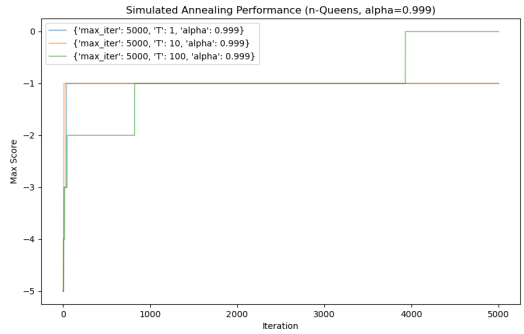
B. Traveling Salesperson Problem

In Figure. 4, Figure. 5, and Figure. 6 we can find the results for the Traveling Salesperson problem. Our observations are that:

- In general, all three algorithms get stuck in some local optimum with a smaller number of iterations. But overall, the genetic algorithm performs the best among all three algorithms.
- The genetic algorithm performs better possibly because it takes larger movements when exploring the space during the crossover step, while the other two algorithms take much more granular steps.



(a) Simulated Annealing - α = 0.99



(b) Simulated Annealing - α = 0.999

Fig. 2. The scores for 8-Queens problem from simulated annealing algorithm. An optimal solution should have score = 0.

C. Neural Network Optimization

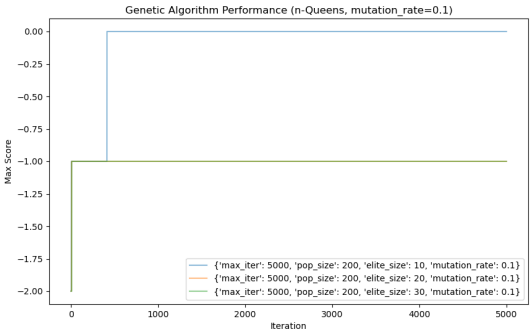
In Figure. 7, Figure. 8, and Figure. 9 we can find the results for the neural network optimization.

- The genetic algorithm has the overall best performance among the three optimization algorithms.
- The simulated annealing algorithm is most likely to get stuck in local optima.
- The random hill climbing algorithm moves the slowest towards better states.
- A possible explanation is that we are now facing an unbounded and much higher-dimensional optimization problem (in total 224 dimensions with range in real numbers), and hence the more aggressive genetic algorithm performs better.

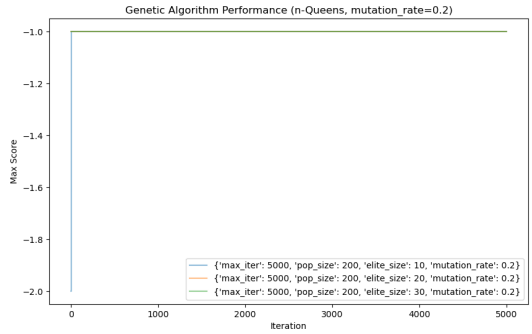
VI. Discussion

Our experiment results confirm our hypotheses that

1) The hyper-parameters will significantly affect the performance of these optimization algorithms.



(a) Genetic Algorithm - mutation rate = 0.1



(b) Genetic Algorithm - mutation rate = 0.2

Fig. 3. The scores for 8-Queens problem from genetic algorithm. An optimal solution should have score = 0.

2) There will not be a single optimization algorithm that outperforms all others.

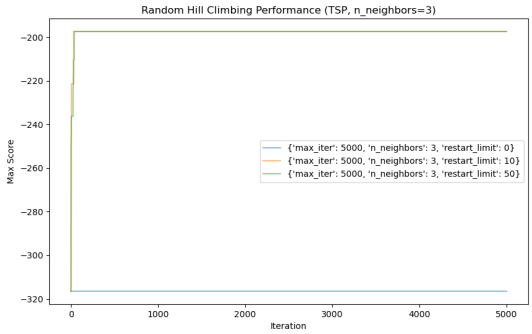
See more detailed discussion below.

A. Hyper-parameters are crucial to the performance of optimization algorithms

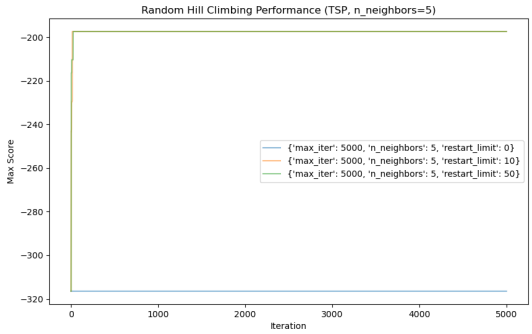
Our biggest finding is that hyper-parameters could be the most crucial factor to the performance of a given optimization algorithm. However, not all hyper-parameters contribute uniformly. An obvious example could be when we are applying the random hill climbing algorithm to the 8-city traveling salesperson problem. While the number of neighbors does not really affect the final results, we could see that a high number of restart limits will improve the algorithm performance by at least 50%.

B. All optimization algorithms fail in some way, but some are useful

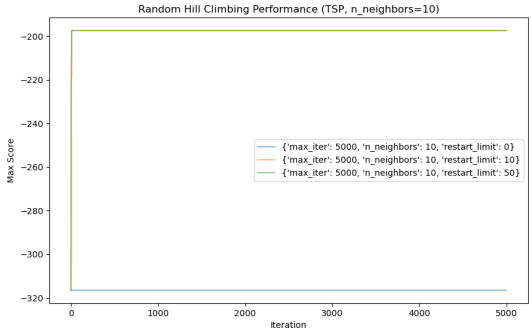
We also find that even though we are only considering three optimization problems and three optimization algorithms here, the best algorithm for each type of problem varies. Problems defined in a smaller space would prefer an algorithm that makes more granular searching steps, while NP-hard types of problems or problems defined in higher-dimensional spaces would benefit more from larger searching steps.



(a) Random Hill Climbing - neighbor=3



(b) Random Hill Climbing - neighbor=5



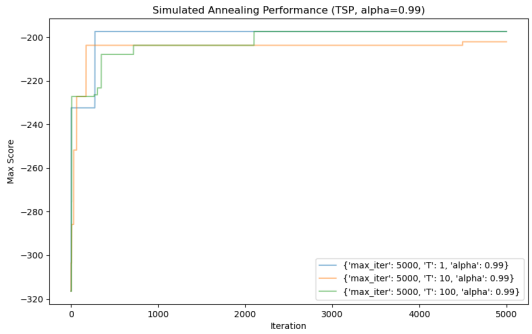
(c) Random Hill Climbing - neighbor=10

Fig. 4. The scores for 8-city traveling salesperson problem from random hill climbing algorithm. The larger the score is, the less the traveling distance is.

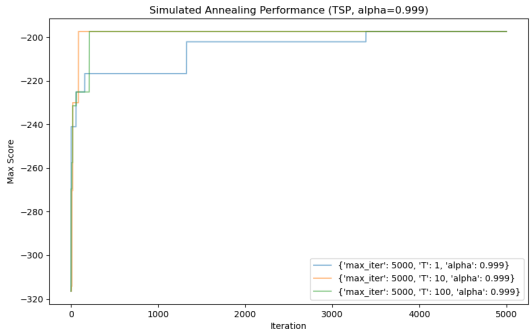
C. Future Study

In our next step, we would like to:

- Conduct a more thorough and careful hyper-parameter tuning. In this report, due to time constraints, we only selected a few hyper-parameters and tested for a few discrete values. A more detailed hyper-parameter tuning would help us to better understand how hyper-parameters affect the performance of each type of optimization algorithm.
- Compare the above algorithms with MIMIC [**?**]. Due to time limits, we didn't implement the MIMIC algorithm. Adding MIMIC to the experiments will help us to better



(a) Simulated Annealing - α = 0.99



(b) Simulated Annealing - α = 0.999

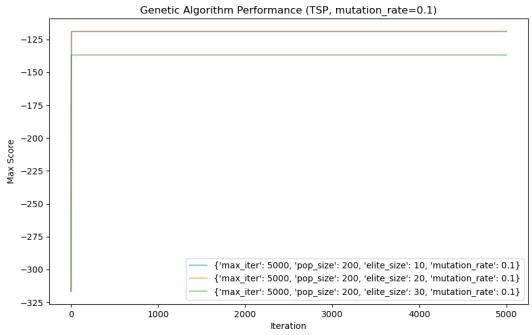
Fig. 5. The scores for 8-city traveling salesperson problem from simulated annealing algorithm. The larger the score is, the less the traveling distance is.

understand the pros and cons of each type of optimization algorithm.

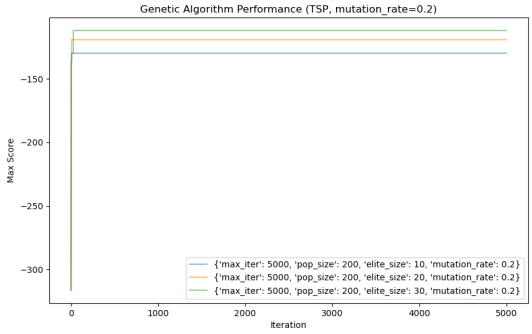
REFERENCES

[1] Hofmann, Hans. (1994). Statlog (German Credit Data). UCI Machine Learning Repository. https://doi.org/10.24432/C5NC77.

[2]

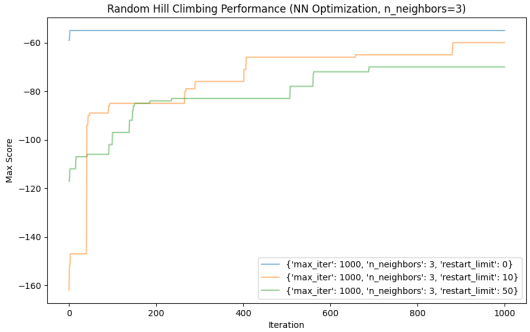


(a) Genetic Algorithm - mutation rate = 0.1

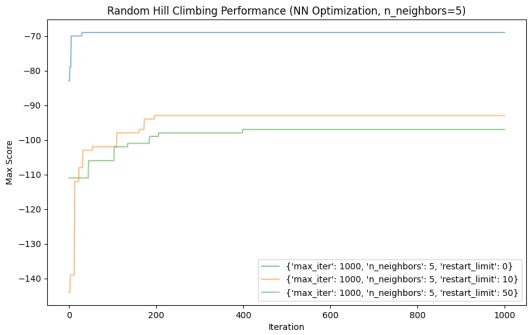


(b) Genetic Algorithm - mutation rate = 0.2

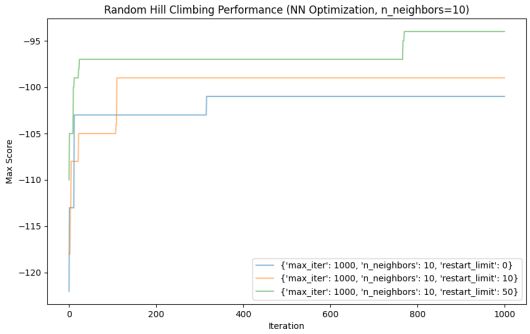
Fig. 6. The scores for 8-city traveling salesperson problem from genetic algorithm. The larger the score is, the less the traveling distance is.



(a) Random Hill Climbing - neighbor=3

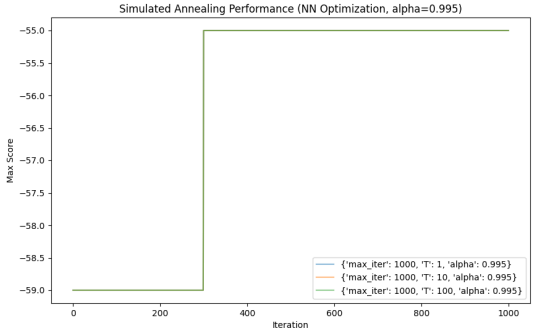


(b) Random Hill Climbing - neighbor=5

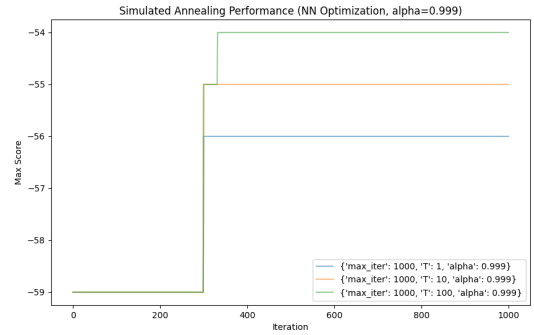


(c) Random Hill Climbing - neighbor=10

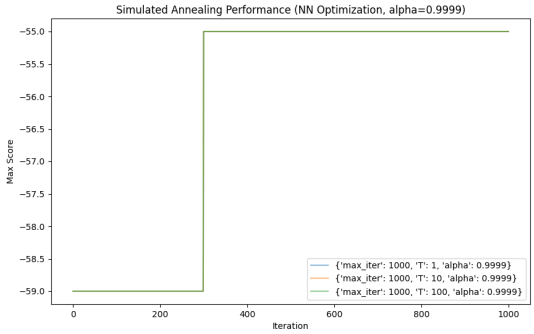
Fig. 7. The scores for the fitted neural network from random hill climbing algorithm. The larger the score is, the lower the weighted loss is.



(a) Simulated Annealing - α = 0.995

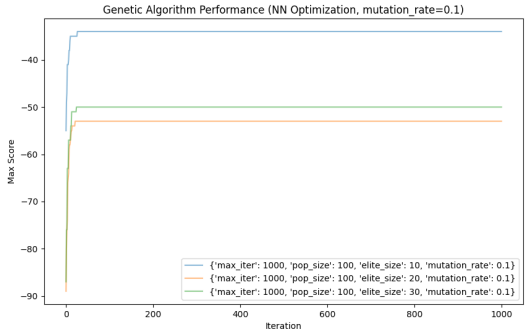


(b) Simulated Annealing - α = 0.999

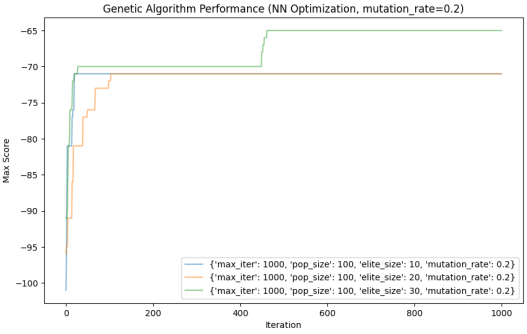


(c) Simulated Annealing - α = 0.9999

Fig. 8. The scores for the fitted neural network from simulated annealing algorithm. The larger the score is, the lower the weighted loss is.



(a) Genetic Algorithm - mutation rate = 0.1



(b) Genetic Algorithm - mutation rate = 0.2

Fig. 9. The scores for the fitted neural network from genetic algorithm. The larger the score is, the lower the weighted loss is.