Report of Assignment 2 - Randomized Optimization

Qizhi Zhao

Georgia Institute of Technologies

***Abstract*—This project explores various optimization techniques in random search problems and neural network weight optimization problem through experiments and comparison analysis. For the two random search problems, I had selected n-queen problem and Traveling Salesman by carefully consideration of the analysing for the current project and the exploring the knowledge. I compare trough these two problem and analyzed to understand the behavior of different algorithms and hyper parameter tuning under different aspects such as learning fitness curve and time performance. This report presents findings from experiments conducted using four optimization algorithms: Random Hill Climbing, Simulating Annealing, Generic Algorithm and MIMIC. I will do several experiment, analysis and discusses the obtained results, compares the algorithms and problems, and suggests potential improvements. This report will emphasizes the strengthens and weakness of such algorithms and the way to improve the performance and avoid the the weakness of it by such techniques.**

***Index Terms*—Machine Learning, Neural Network, Supervised Learning, Randomized Optimization, Weight Optimization, Random Hill Climbing, Simulating Annealing, Generic Algorithm, MIMIC, N-Queen,Traveling Salesman**

I. Introduction

Random optimization algorithms are particularly suitable for solving problems influenced by multiple variables, where there are many possible solutions, and the results vary greatly due to the combinations of these variables. A typical application scenario is when the solution space is so large that it is impossible to try all possible solutions one by one. For such problems, the simplest and least efficient solving method is to attempt randomly guessing thousands of solutions and finding the best one among them. A more efficient approach is to start from a random solution and make adjustments in the direction of possible improvements to approximate the optimal solution. This paper will start with the least efficient random search algorithm and then introduce more optimal random optimization algorithms such as hill climbing, simulated annealing, genetic algorithms, and MIMIC.

II. Methodology

In this section, I will outline the three primary Randomized Optimization algorithms utilized for random searching tasks. By using the mlrose and scikit-learn (sklearn) library in Python, I'll demonstrate the implementation and exploration of parameters and methods, along with the analysis of performance results derived from the random searching problem. This exploration aims to explain how each algorithm get the correct solutions through the random search and discern optimal strategies for employing them in various scenarios.

A. Random Hill Climbing

Hill Climbing is a simple and intuitive optimization algorithm used for solving various problems. It is a local search algorithm that iteratively moves towards improving the objective function value, aiming to find the peak (maximum) or valley (minimum) in the solution space. But the main issue of this algorithm is that it is easy to catch the local maximum, that is, a certain node will be higher than any of its neighbors, but it is only a local optimal solution, not a global optimal solution.To solve this issue , the concept of randomness was introduced.Random Hill Climbing(RHC) iteratively hill climbing, starting from a random initial guess each time, and moving in the direction of increasing fitness in each iteration. This lower the chance to fall into the local minima, but still possible that it will not the most optimal solution. Here is the algorithm step:

1) **Initialization**: Start with an initial Randomdized solution.
2) **Evaluation**: Evaluate the fitness of the current solution using the objective function.
3) **Neighbor Generation**: Generate a neighboring solution by making a random modification to the current solution.
4) **Acceptance Criteria**: Compare the fitness of the neighboring solution to the current solution. If the neighboring solution has a better fitness, accept it as the new current solution.
5) **Iteration**: Repeat the evaluation, neighbor generation, and acceptance steps until a stopping criterion is met (e.g., a maximum number of iterations or a satisfactory fitness level).

B. Simulated Annealing

The simulated annealing algorithm was developed in the early 1980s, drawing its inspiration from the annealing process of solids. This process involves heating a solid to a sufficiently high temperature and then slowly cooling it. During heating, the particles inside the solid become disordered with increasing temperature, leading to an increase in internal energy. When cooled slowly, the particles gradually become more ordered. Theoretically, if the cooling process is slow enough, the solid can reach thermal equilibrium at any temperature during cooling, and at low temperatures, it will achieve the minimum internal energy state at that low temperature.

An important step in this simulated annealing process is achieving thermal equilibrium at any constant temperature. This also can be simulated using the Monte Carlo algorithm. But it requires extensive sampling and it require alot of computing power. However, since physical systems tend to minimize energy, and molecular thermal motion tends to disrupt this low-energy state, it is sufficient to focus on states that contribute significantly. In the year of 1953, Metropolis proposed an importance sampling method. The method involves generating a new state j from the current state i. If the internal energy of the new state j (E_j) is less than that of state i (E_i), then the new state j is accepted as the new current state. Otherwise, state j is accepted with a certain probability P, where

$$P = \exp\left(-\frac{E_j - E_i}{kT}\right)$$

and k is the Boltzmann constant. This is known as the Metropolis criterion.

In 1953, Kirkpatrick drew an analogy between the concept of simulated annealing and combinatorial optimization, applying simulated annealing to combinatorial optimization problems. When applying the simulated annealing algorithm to optimization problems, the temperature T is typically treated as a control parameter, and the objective function value f is regarded as the internal energy E. A state of the solid at a given temperature T corresponds to a solution. The algorithm then attempts to lower the objective function value f (internal energy E) as the control parameter T decreases, aiming to reach the global minimum (the lowest energy state at low temperature during annealing), similar to the solid annealing process.Here is the algorithm step:

1) **Initialization**: Start with an initial solution and an initial temperature T.
2) **Perturbation**: Generate a neighbor solution by making a small change to the current solution.
3) **Acceptance Criteria**: Evaluate the change in the fitness value, $\Delta E = E_{\text{new}} - E_{\text{current}}$. Accept the new solution if it improves the objective function (i.e., $\Delta E < 0$). If $\Delta E \geq 0$, accept the new solution with a probability $P = \exp\left(-\frac{\Delta E}{kT}\right)$, where k is the Boltzmann constant.
4) **Cooling**: Reduce the temperature T according to a predefined cooling schedule (e.g., $T \leftarrow \alpha T$, where $\alpha < 1$).
5) **Iteration**: Repeat the perturbation, acceptance, and cooling steps until the system reaches convergence criteria such as thermal equilibrium or the temperature is sufficiently low.

C. Genetic Algorithm

Genetic Algorithm (GA) is a algorithm that simulate the biological evolution process based on the principles of Darwinian natural selection and genetic mechanisms. It is a method for finding optimal solutions by simulating the natural evolution process.

The main features of GA include its ability to directly manipulate structural objects without assumptions of derivatives or function continuity. It inherently has parallelism and strong global optimization capabilities. GA adopts a probabilistic optimization method, which enables it to autonomously explore and guide the search space without the need for deterministic rules, thereby adaptively adjusting its search direction.Here is the algorithm step:

1) **Initial Population Generation**: Begin with a randomly generated initial population of candidate solutions.
2) **Fitness Evaluation**: Evaluate the fitness of each candidate solution within the population.
3) **Selection**: Select the fittest individuals to reproduce, based on their fitness values.
4) **Crossover**: Perform crossover operations to combine pairs of selected individuals (parents) to produce offspring. This step simulates genetic cross-over.
5) **Mutation**: Apply mutation operations to some individuals in the population to introduce variability. This step simulates genetic mutation.
6) **Iteration**: Repeat the fitness evaluation, selection, crossover, and mutation steps until convergence criteria are met (e.g., a maximum number of generations or a satisfactory fitness level).

D. MIMIC

The Mutual-Information-Maximizing Input Clustering (MIMIC) algorithm is a probabilistic model-building genetic algorithm (PMBGA) designed to optimize complex problems. Unlike traditional genetic algorithms, MIMIC constructs a probabilistic model of provide solutions and uses this model to guide the search process.Here is the algorithm step:

1) **Initial Population Generation**: Start with a randomly generated initial population of candidate solutions.
2) **Fitness Evaluation**: Evaluate the fitness of each candidate solution in the population.
3) **Modeling**: Construct a probabilistic model based on the current best solutions. This model included estimating the joint probability distribution of the variables.
4) **Sampling**: Use the probabilistic model to sample new candidate solutions. These samples are more likely to be close to the high-fitness regions identified by the model.
5) **Iteration**: Repeat the fitness evaluation, model building, and sampling steps until convergence criteria are met

III. Problems

A. N-Queen Problem

1) Description: The problem is to safely placing n queens on an $n \times n$ chessboard. "Safely" means that no two queens can occupy the same row, column, or diagonal, ensuring that none of the queens can attack each other. This problem originates from the "Eight Queens Problem," first proposed by chess player Max Bezzel in 1848. The first solution was provided by Franz Nauck in 1850, who also extended the problem to the more general case of placing n queens on an $n \times n$ board. Nauck's work laid the foundation for the generalized n-queens placement problem, which remains a significant challenge in optimization and theoretical computer science.

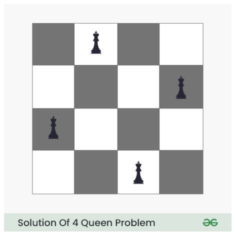

Fig. 1. N-Queen Problem

B. Traveling Salesman Problem

1) Description: The problem suppose that there is a traveling Salesman who wants to visit n cities. The TSP can be stated as follows: Given a list of n cities and the distances between each pair of cities, the purpose for the question is to find the shortest possible route that visits each city exactly once and returns to the starting city. The TSP is an NP-hard problem. This means that there is no known polynomial-time algorithm to solve all instances of the problem optimally. For large instances, exact algorithms become impractical, and heuristic or approximation algorithms are used.

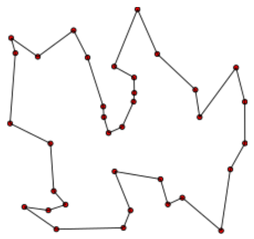

Fig. 2. Traveling Salesman Problem

C. Thought about these two problems

I chose the N-Queen and Traveling Salesman Problems (TSP) for my project because they are classic optimization problems that offer valuable insights into the performance of various optimization algorithms. The N-Queen problem is a well-known challenge in the field of artificial intelligence for analyzing the effectiveness of different algorithms in finding optimal or near-optimal solutions in a constrained search space. On the other hand, the TSP is a fundamental problem in operations research and computer science, known for its complexity and applicability to real-world scenarios such as route planning and logistics. By comparing the performance of algorithms like Random Hill Climbing, Simulated Annealing, Genetic Algorithms, MIMIC on these problems, I aim to understand their strengths, weaknesses, and suitability for different types of optimization tasks, thereby gaining a deeper understanding of their practical applications and limitations in neural network training.

IV. Experiment

A. All algorithms in two problem

1) Description: For this experiment, I will run all four algorithms on two problems using the same parameters: a maximum of 100 attempts, a maximum of 300 iterations, and 20 runs per algorithm. I will collect the mean and standard deviation of the fitness function values and plot them on a graph. For the Traveling Salesman Problem, I will use 10 cities (N=10), and for the N-Queen Problem, I will use 5 queens (N=5). The focus of this experiment is not on the problem size, but rather on comparing the time cost and behavior of each algorithm.

2) Result and Discussion: First, let's discuss the N-Queen Problem (with N =5) that shows in Figure 3 and Table I. In this problem, the fitness function measures the number of conflicts (how many queens violate the attack rule). The goal is to minimize the fitness function value to 0.

For Random Hill Climbing (RHC), as expected, it has the fastest speed among all four algorithms. However, it has the potential to fall into local minima. The mean curve of 20 runs for the RHC algorithm is at 0.45. This indicates that sometimes RHC stops in a local minimum (fitness value of 1) and cannot reach the optimal solution (fitness value of 0). The standard deviation boundaries remain about the same and do not get closer, indicating that the likelihood of falling into a local minimum does not decrease regardless of the number of iterations.

For Simulated Annealing (SA), the mean curve of the fitness values reaches 0 around iteration 175, and the standard deviation boundaries approach 0. This indicates that the simulated annealing technique can escape from local minima and find the global solution. Among the algorithms that can find the global solution, SA runs the fastest.

For the Genetic Algorithm (GA) and MIMIC, both are fast convergence algorithms and can always find the global best solution within the first five iterations. However, their speed per iteration is much slower than Random Hill Climbing or Simulated Annealing because each iteration requires evaluating a population of solutions.

Next, we discuss the Traveling Salesman Problem (TSP) (with City N = 10) shows in the Figure 4 and Table II. In this problem, the fitness function is the total distance of the path that the traveler needs to complete. This is also a minimization problem; we aim to minimize the fitness function value. In this experiment, we observe more clearly how the algorithms perform differently due to the larger problem size, which makes the problem more challenging.

For Random Hill Climbing, the performance is similar to what we saw in the N-Queen Problem. The mean fitness value converges to the highest value compared to all other algorithms, indicating that it does not converge to the global solution but falls into local minima. The upper and lower bounds of the fitness values remain the same, suggesting that performance does not improve even with more iterations.

For Simulated Annealing, it shows that it cannot guarantee a global solution for larger problem sizes.

For Genetic Algorithm, it achieves the global optimal solution within 50 iterations, and most of the time, it finds the most optimal solution because the standard deviation of 20 runs is 0.16.

For MIMIC, it does not perform as well as it does in the N-Queen Problem. This is because MIMIC relies on a probability model to generate candidate solutions, which might cause it to frequently fall into local minima.

3) Conclusion: Random Hill Climbing (RHC) is the fastest in terms of run time for each iteration but it is getting stuck in local minima. It often fails to consistently find the global solution, and its performance does not improve with more iterations. The standard deviation remains constant, indicating persistent issues with local minima.

Simulated Annealing (SA) can escape local minima and find the global solution. The mean fitness values reach and almost reach the optimal solution over time, and the decreasing standard deviation shows improved performance with more iterations. However, SA has very high possibility but may not guarantee global solutions for larger, more complex problems.

Genetic Algorithm (GA) always finds the global best solution within a few iterations, making it effective for both small and large problem sizes. However, it is computationally expensive per iteration due to the need to evaluate a population of solutions, resulting in slower per-iteration speed compared to RHC and SA.

MIMIC shows fast convergence in simpler problems like the N-Queen (when N=5) Problem, effectively finding global solutions quickly. However, its performance declines in more complex problems like the Traveling Salesman Problem due to its reliance on probability models, leading to frequent convergence to local minima.

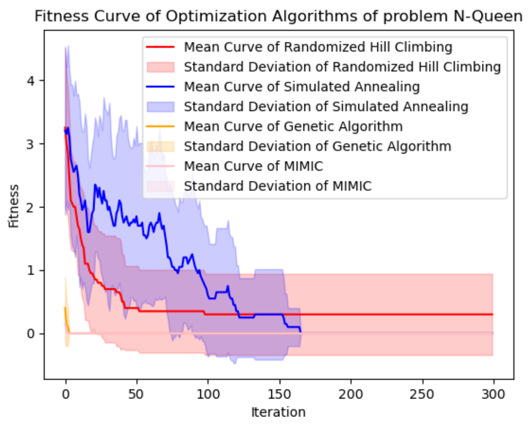

Fig. 3. Fitness Curve of Optimization Algorithms of N-Queen Problem

B. Problem Size

1) Description: From the previous experiment, we found that the problem has an impact on the algorithms, but we do not know how it affects the performance of those four algorithms.

Therefore, I ran all four algorithms with problem sizes ranging from 5 to 105, with a step size of 5, and recorded their runtime and fitness values. From this experiment, we can visually observe the relationship between the problem size and the runtime and fitness of the algorithms.

2) Result and Discussion: On Figure5, the graph of showing the relationship between problem size and fitness value(average of 5 runs) for the four different optimization algorithms Travelling Salesman Problem.We can see that the Fitness value will increase linearly for the problem size increasing for all four algorithm. And as I had mention on the Experiment 1, MIMIC will always converge to the local minimum and though it was the right answer. Random Hill Climbing and Simulated Annealing has similar fitness return as the mean fitness value. But barely we can see that the Simulated Annealing did little bit better as the problem size growth after problem size 50 from the best fit of both. The Genetic Algorithm performs the best to get the lowest fitness among four algorithms.

On Figure 6, the graph shows the relationship between problem size and run time for the four different optimization algorithms Travelling Salesman Problem.MIMIC is the special

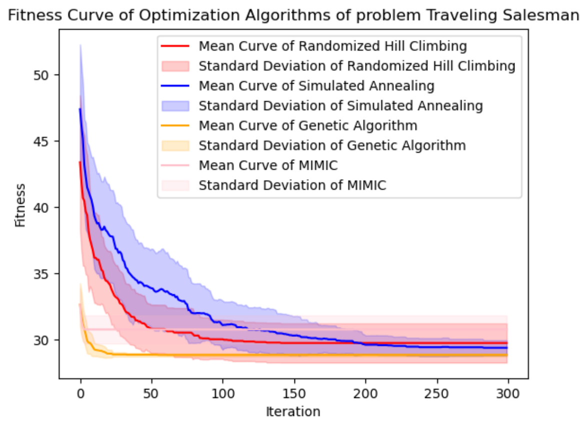

Fig. 4. Fitness Curve of Optimization Algorithm of Traveling Salesman Problem

TABLE I
COMPARISON OF RUN TIME 5-QUEEN PROBLEM

	RHC	SA	GA	MIMIC
Loss-Mean	0.45	0	0	0
Loss-Std	0.80	0	0	0
Runtime(s) Average of 20runs	0.0207	0.0309	3.8943	3.9822

TABLE II
COMPARISON OF RUN TIME 10 CITY TRAVELING SALESMAN PROBLEM

	RHC	SA	GA	MIMIC
Loss-Mean	29.75	29.39	28.84	30.77
Loss-Std	1.46	1.53	0.11	1.055
Runtime(s) Average of 20runs	0.0200	0.0346	4.1230	14.7661

one, while the problem size increased, the average runtime increase quadratics.By my understanding, many operations in MIMIC involve matrix manipulations, such as calculating mutual information between variables. The time complexity of these operations can be quadratic respect to the number of variables which is problem size in this case. For all the other algorithm, we have Figure 7 to have a better view of the lines. we can see that runtime of each algorithm increase linearly respect to the problem size.Also from the same graph, we can clearly see that the run time of the Genetic Algorithm is much more higher than the run time of Random Hill Climbing and Simulated Annealing.

3) Conclusion: In conclusion, the performance of the four optimization algorithms—Random Hill Climbing (RHC), Simulated Annealing (SA), Genetic Algorithm (GA), and MIMIC—varies significantly based on problem size and computational efficiency.

For the Traveling Salesman Problem, the fitness values for all four algorithms increase linearly with problem size. However, MIMIC consistently converges to local minima, leading to local solutions. RHC and SA perform similarly, but SA shows a slight improvement as problem size exceeds 50. The GA achieves the best fitness values, consistently finding the lowest values among the four algorithms.

Regarding runtime, MIMIC's average runtime increases quadratically with problem size due to its reliance on matrix manipulations, making it less efficient for larger problems. In contrast, the runtimes of RHC, SA, and GA increase linearly with problem size. Despite GA's superior performance in finding optimal solutions, its runtime is significantly higher than that of RHC and SA, making it less efficient computationally.

Overall, RHC offers speed but struggles with local minima, making it less reliable for finding global solutions. SA balances speed and reliability, particularly effective for moderate problem sizes. GA is robust and reliable, consistently finding optimal solutions but at a higher computational cost per iteration. MIMIC excels in simpler problems but struggles with complexity, highlighting its dependency on the nature of the problem. Each algorithm has distinct advantages and limitations, making them suitable for different types of optimization problems based on specific requirements and constraints.

C. Neural Network

1) Background Information: For this experiment, I will use the digit dataset from a previous project to train the neural network. Optimization algorithms in neural networks adjust weights to minimize the loss function, thereby improving model accuracy and performance. Common optimization algorithms include stochastic gradient descent (SGD) and Adam (Adaptive Moment Estimation), which are utilized with backpropagation to update weights effectively.

2) Parameter setup: I initially tried using a low maximum iteration count, such as 200, but the model did not converge within that number of iterations. Therefore, I had to increase the max_iters parameter to 10,000. For the hidden layer nodes (the variable in question), I opted for a single layer of 70

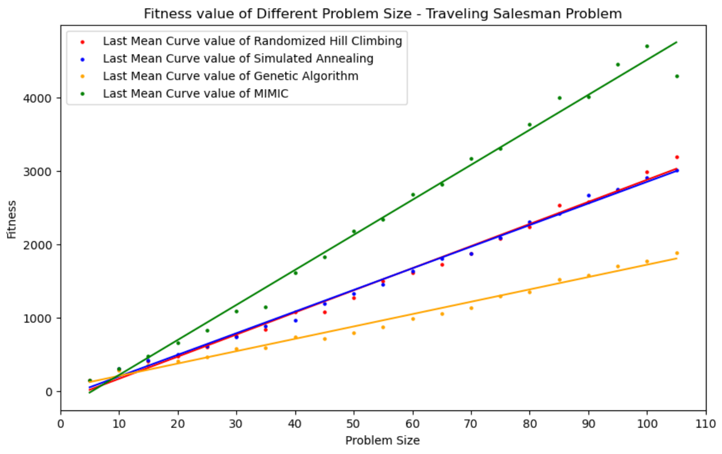

Fig. 5. Fitness Curve of Different problem size

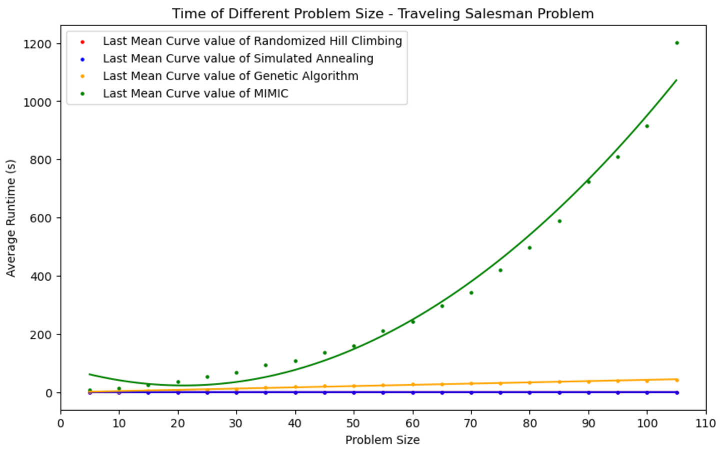

Fig. 6. Time of Different problem size

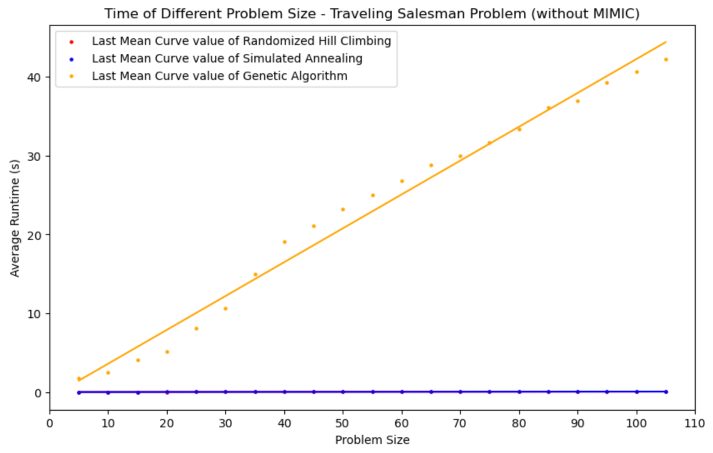

Fig. 7. Time of Different problem size without MIMIC

nodes. This decision was influenced by the fact that the digit dataset's images are 8x8 pixels, and typically, the number of nodes in the hidden layer should exceed the number of inputs. Thus, I chose 70 nodes to optimize computing resources, as training required significant time and computational power.

3) Description: During this project, we explored three optimization algorithms: Random Hill Climbing, Simulated Annealing, and Genetic Algorithm. In this experiment, we will investigate the effects of applying these three optimization algorithms to neural networks. After analyzing and understanding the results, we aim to explain why modern neural network training typically does not use these optimization algorithms and instead relies on SGD (Stochastic Gradient Descent) and Adam.

4) Result and Discussion: For Random Hill Climbing and Simulated Annealing, they perform similarly in terms of loss function values for each iteration, as observed from the curve in Figure 8 depicting the relationship between iteration and loss value. Both algorithms reach a loss value of 251.80 and 243.04 respectively by the 10,000th iteration. However, when compared to the loss value of 0.0020 achieved by Adam (as shown in Table III), the losses for these two algorithms are significantly higher.

For Genetic Algorithm, it have very similar convergence graph (Figure 9) with adam(Figure 10), but the iteration number are way bigger. Although it requires a longer training time of about 2 hours (6788.34 seconds), it achieves a lower loss value compared to both Random Hill Climbing and Simulated Annealing. However, when compared to Adam, the Genetic Algorithm's performance is still not sufficient, and its computational time is excessively high.

For Adam, it achieves a training time of 0.9119 seconds and a loss value of 0.0020, making it the fastest and most effective optimizer among the four. Its speed and performance can be attributed to its adaptive learning rate and momentum mechanisms. Adam adjusts learning rates based on first and second moment estimates of gradients, allowing it to navigate gradient landscapes efficiently and accelerate convergence.

TABLE III
COMPARISON OF RUN TIME 10 CITY TRAVELING SALESMAN PROBLEM

	RHC	SA	GA	ADAM
Loss Value	251.80	243.04	3.2962	0.0020
Training Time(s)	25.38	29.45	6788.34	0.9119

REFERENCES

[1] Tom Mitchell, McGraw Hill. Machine Learning,McGraw-Hill Science/Engineering/Math,1997.

[2] Zhou Zhi-Hua. Machine Learning[M]. Tsinghua University Publishing, 2016.

[3] Hecht-Nielsen, Robert. "Theory of the backpropagation neural network." Neural networks for perception. Academic Press, 1992. 65-93.

[4] Reddi, Sashank J., Satyen Kale, and Sanjiv Kumar. "On the convergence of adam and beyond." arXiv preprint arXiv:1904.09237 (2019).

[5] Rivin, Igor, Ilan Vardi, and Paul Zimmermann. "The n-queens problem." The American Mathematical Monthly 101.7 (1994): 629-639.

[6] Hoffman, Karla L., Manfred Padberg, and Giovanni Rinaldi. "Traveling salesman problem." Encyclopedia of operations research and management science 1 (2013): 1573-1578.

[7] Mirjalili, Seyedali, and Seyedali Mirjalili. "Genetic algorithm." Evolutionary algorithms and neural networks: Theory and applications (2019): 43-55.

[8] Van Laarhoven, Peter JM, et al. Simulated annealing. Springer Netherlands, 1987.

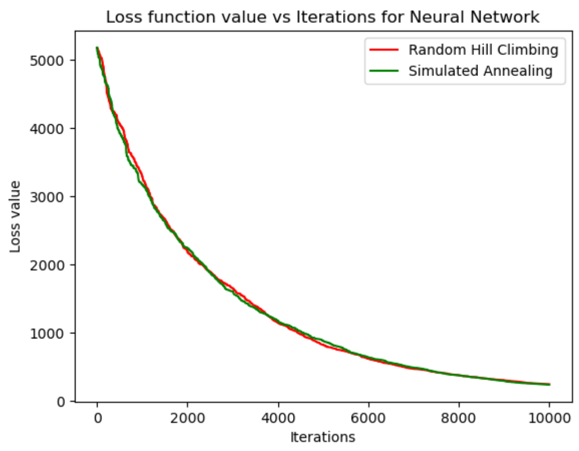

Fig. 8. Random Hill Climbing and Simulated Annealing Loss Curve - Neural Network

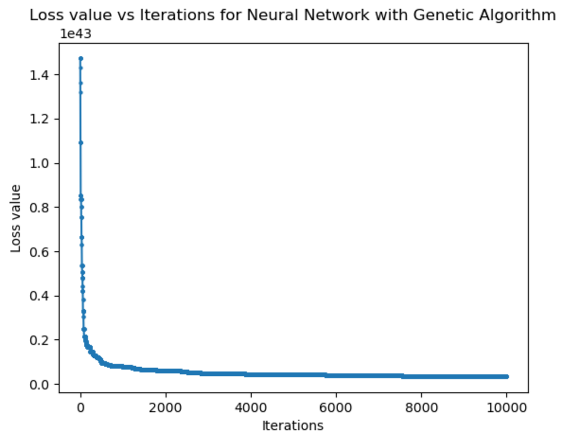

Fig. 9. TGenetic Algorithm Loss Curve - Neural Network

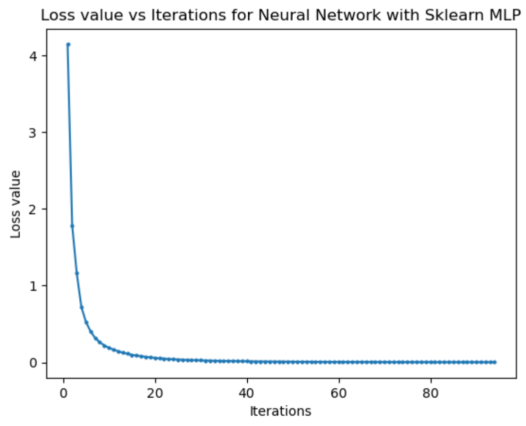

Fig. 10. Default MLP optimizer (Adam)