CS7641 Assignment 2: Randomized Optimization

Boyuan Liu
bryanliu@gatech.edu

1. Introduction

Randomized optimization is an important unsupervised learning method that can solve a wide variety of problems. In this assignment, I explored four randomized optimization algorithms: randomized hill climbing, simulated annealing, genetic algorithm, and MIMIC. These algorithms were implemented and compared in three classic optimization problems: 8-queen, flip flop, and six peaks. They are also used to optimize weights in the neural network from the previous assignment and compared to the gradient descent algorithm.

2. Methods

2.1 Randomized Hill Climbing

The randomized hill climbing algorithm builds on top of the regular hill climbing algorithm. The regular hill climbing algorithm starts at a random location and climbs towards the peak with the highest slope. The primary issue of this algorithm is that the optimizer is often stuck at local optima. To address this issue, the randomized hill climbing algorithm performs random restarts at different locations to find the global optima. This technique greatly increases the chance of finding the global optima at the cost of an increased number of function evaluations.

2.2 Simulated Annealing

Similar to the randomized hill climbing algorithm, the simulated annealing algorithm aims to "climb" to the global optima. However, instead of always "climbing" up the hill, the simulated annealing algorithm has a chance to jump to a new sample when the new step is not as good as the previous one. This probability is defined by the following equation, where T is the temperature. The behavior of this algorithm is controlled by the temperature T. When the temperature is near zero, this algorithm behaves similarly to hill climbing, and when the temperature is very high, this algorithm behaves more like random walk. This mechanism enables the algorithm to better explore the input space, which often leads to better performance.

$$P = \begin{cases} 1 & if \Delta c \leq 0 \\ e^{-\Delta c / t} & if \Delta c > 0 \end{cases}$$

2.3 Genetic Algorithm

The genetic algorithm took inspiration from the process of natural selection and genome crossover, hence the name "genetic". The algorithm usually starts with a fixed population size and evaluates the fitness of each individual. The "fittest" individual among the population would be selected and paired. A new generation of individuals will then be generated by crossing over the fittest among the previous generation. Like natural selection, this process would ultimately lead to more optimized results.

2.4 MIMIC

The previously mentioned algorithms have various issues, such as the lack of overall structure and unclear probability distribution. The MIMIC algorithm, on the other hand, directly models probability distribution and conveys structure. It uses the generated probability distribution of all samples and sets the new threshold θ to the nth percentile. It will then generate a new probability distribution over the newly selected samples. The algorithm will repeat this process until the threshold reaches θ_{max}, which is the optima. The remaining samples will be the optimal results.

3. Optimization Problems

3.1 The 8-Queen Problem

The 8-queen problem is a simple optimization problem of chess pieces. The queen pieces can attack other pieces vertically, horizontally, and diagonally. The goal of this problem is to place eight queen pieces on a chessboard in a way that none of them can attack one other. The eight queens are placed diagonally initially, and only their column positions need to be optimized. Therefore, this problem has an input size of eight, and the maximum value is also eight. The fitness function is defined as how many pairs of queens cannot attack each other. Therefore, the highest fitness score is 28. The convergence condition is set to be when the highest fitness score does not improve after a max attempt is reached. I first explored how the number of maximum attempts affects the performance of these algorithms. I varied the number of maximum attempts from 1 to 30 while keeping the other parameters as default. As shown in Figure 1, the optimizers' best fitness scores increase with the number of maximum attempts, except for MIMIC. This is because the optimizer sometimes gets stuck at local optima Having more attempts is more likely to find the global optima. Increasing the max attempts also leads to a higher

number of iterations and function evaluations, as shown in the figure, which means higher computation time. I chose 15, 30, 30, and 2 max attempts for the four algorithms, respectively, after balancing the performance and cost.

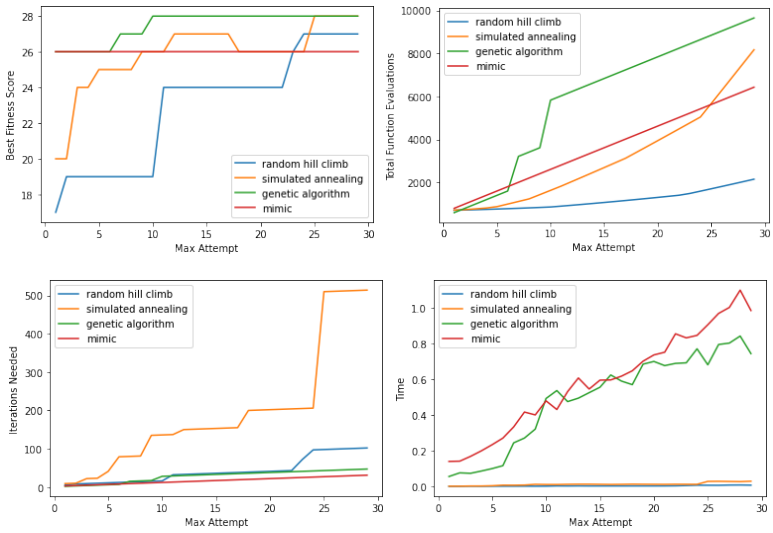
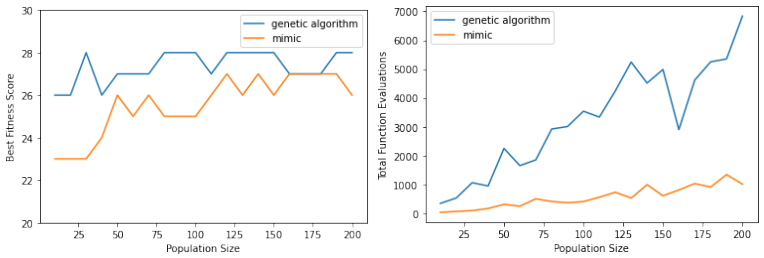


Figure 1 - The impact of maximum attempts on optimizer performance.

Both the genetic algorithm and the MIMIC algorithm use a population size parameter. The generic algorithm starts with a fixed population size and selects the fittest individuals among them. The MIMIC algorithm generates a probability distribution of all populations and selects the nth percentile using a threshold θ. As shown in Figure 2, increasing the population size leads to a slight increase in fitness scores. This is because a larger population size can provide the optimizer with more samples to select from at each step, which means a higher probability of including the optimal solution at each step. The increase in population size also leads to a higher number of function evaluations and longer computing time, especially for the genetic algorithm. I chose an 80 population size for the generic algorithm and a 175 population size for the MIMIC algorithm.



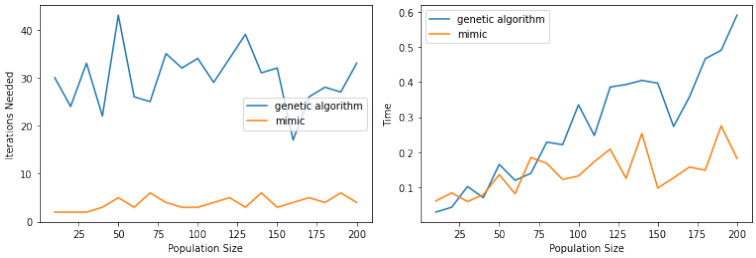


Figure 2 - The impact of population size on optimizer performance.

The generic algorithm uses a mutation probability to generate mutations at the state vector during reproduction, which makes the optimizer explore the state space. The MIMIC algorithm uses a keep ratio to determine how many samples are kept at each iteration. The impact of these two parameters on the fitness score is shown in Figure 3. A 10% probability was adopted for both algorithms. The simulated annealing algorithm uses a temperature T to control the exploration/exploitation ratio of the algorithm. As shown in the third graph in Figure 3, the fitness score shows a general downward trend as the temperature increases. This is because the given 8-queen problem does not need too much exploration to reach the optimal result.

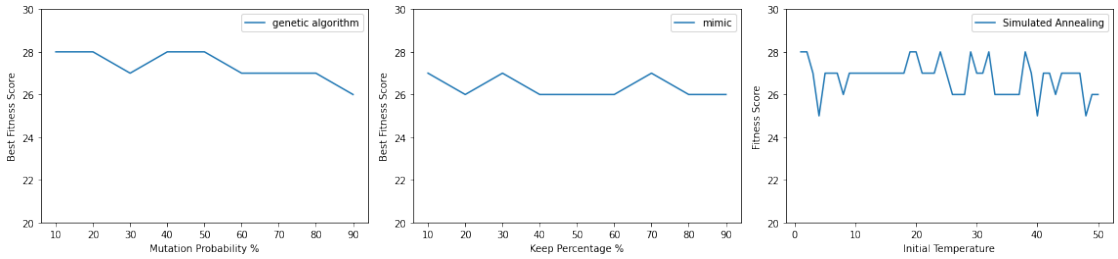


Figure 3 - The impact of mutation probability and keep ratio on the fitness score.

The final optimizers are evaluated after hyperparameter tuning. The results of the fine-tuned optimizers for the 8-queens problem are shown in Figure 4. The genetic algorithm and simulated annealing reached the highest fitness score of 28, and the randomized hill climbing and MIMIC converged at 27. The four optimizers took 0.006s, 0.029s, 0.285s, and 0.134s to solve the problem, respectively.

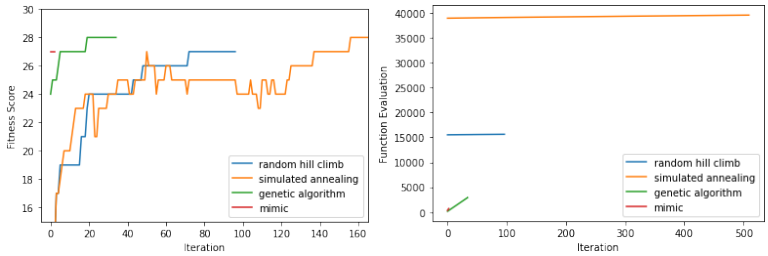


Figure 4 - The optimizer performance on the 8-queen problem after hyperparameter tuning.

After comparing the four algorithms on the 8-queen problem, I further explored their potential for increased problem size, which is essentially an n-queen problem. I explored the state space from 4 to 22 and evaluated the algorithms using the same metrics as above. I used a fitness score percentage (fitness score/max possible fitness score) as the primary metric because the regular fitness score increases with problem size. As shown in Figure 5, the total number of function evaluations and computing time increases with problem size, especially for the genetic algorithm. All four algorithms performed very well at the larger problem scale. The generic algorithm has the best overall performance since it has the highest fitness score percentage across almost the entire problem size space.

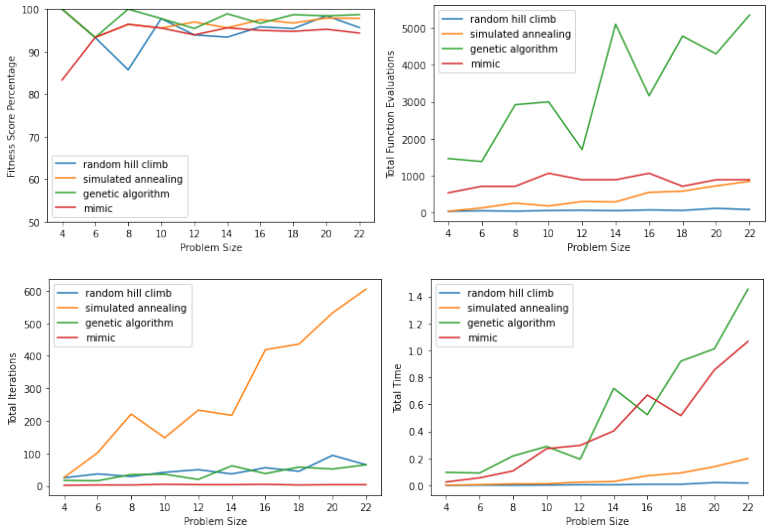


Figure 5 - Optimizer performance on different problem sizes.

3.2 The Flip Flop Problem

The flip flop problem is a simple optimization task that aims to maximize the length of alternative bits within a bit string. For a bit string of size *n*, the maximum fitness score is *n-1*. I used the same hyperparameter tuning methods as the previous problem on a 12-bit flip flop. Some of the plots from hyperparameter tuning are not shown due to page limitations. As shown in Figure6, the generic algorithm and MINIC needed at least a population size of 75 to perform well. The initial temperature for simulated annealing was kept at default value because a higher temperature showed a tendency of instability. The mutation probability and keep ratio were kept at 0.1 based on their fitness score.

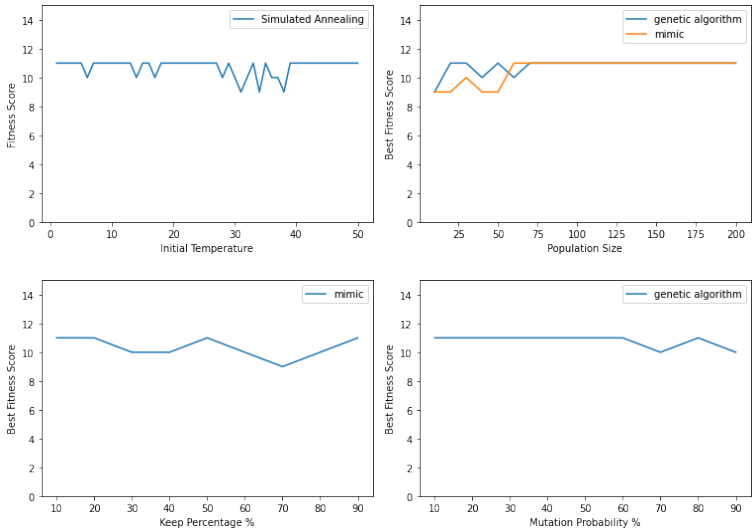


Figure 6 - Hyperparameter tuning results of the flip flop problem.

The results of the fine-tuned optimizers for the 12-bit flip flop problem are shown in Figure 7. The simulated annealing algorithm achieved the highest fitness score and outperformed the other algorithms by a large margin. Although it took significantly more iterations to converge, it only took 0.107 seconds to compute. This is higher than the randomized hill climbing's 0.008 seconds but still better than genetic algorithm's 0.129 seconds and MIMIC's 0.161 seconds.

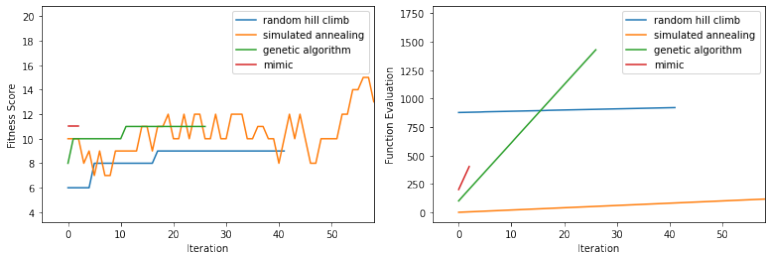


Figure 7 - The fine-tuned optimizers for the 12-bit flip flop problem.

In order to examine the algorithms in a larger state space, I expanded the domain of the problem size. As shown in Figure 8, all four algorithms performed well when the problem size was under 10. Their performances started to drop below the max fitness score as the problem size increased, especially the random hill climbing algorithm. This issue could potentially be resolved by further tuning the optimizers for large problem sizes, such as increasing the population size or increasing the max attempt. The simulated annealing algorithm achieved the highest fitness score throughout the entire problem size space. Although it took the most iterations to converge, it required the second-lowest function

evaluations and hence the second-lowest computing time. Therefore, the simulated annealing algorithm had the best overall performance in the flip flop problem.

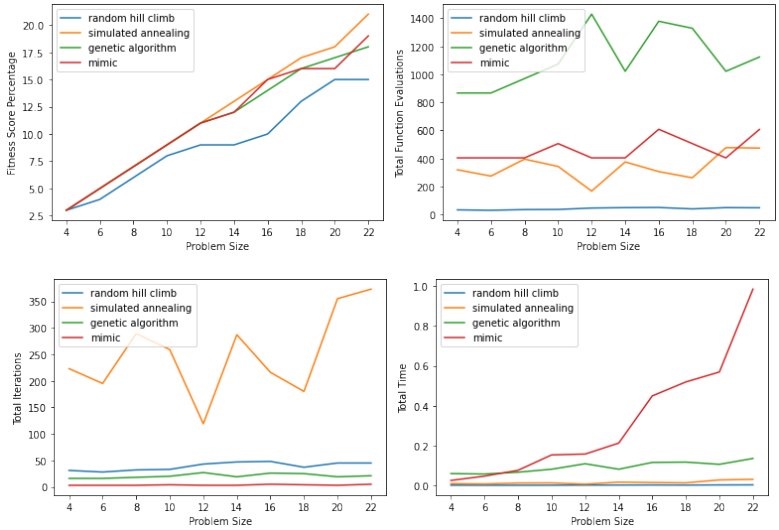


Figure 8 - Optimizer performance on different problem sizes.

3.3 Six Peaks Problem

The six peaks problem is a slightly more complex version of the four peaks problem. It contains two additional global maxima. In this problem, the state vector's structure is more important than its values in order to achieve a high fitness score [1]. I started with a 12-bit six peaks problem and performed hyperparameter tuning. The optimizers' performances fluctuate at lower population sizes and become stable when the population size is over 150. The simulated annealing algorithm is more sensitive to the temperature in this problem than in the previous problems. The optimizers had the best performance at 0.1 mutation probability, and 0.2 keeps percentage, similar to the previous problems.

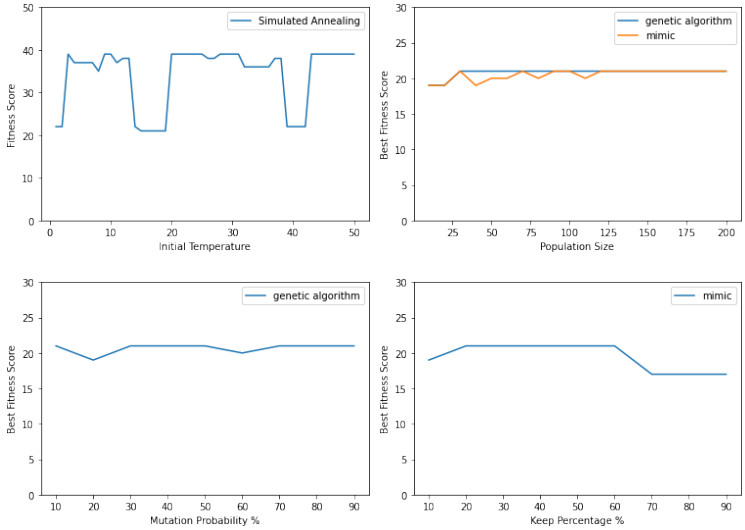


Figure 9 - Hyperparameter tuning results of the six peaks problem.

The results of the fine-tuned optimizers for the 12-bit six peaks problem are shown in Figure 10. All four algorithms achieved high fitness scores, while random hill climbing completed the task in merely 0.001 seconds. Simulated annealing and genetic algorithm were also completed quickly in 0.025s and 0.061s, respectively. MIMIC took 0.151s which was the longest among the four algorithms.

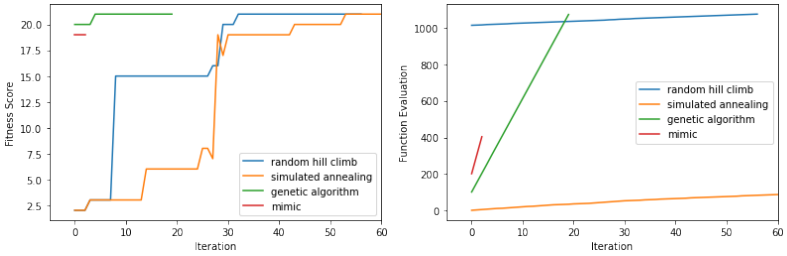


Figure 10 - The fine-tuned optimizers for the 12-bit six peaks problem.

The impact of the problem sizes was studied using the same methods as the previous problems. As shown in figure 11, simulated annealing, genetic algorithm, and MIMIC performed well when the problem size was below 12. However, when the problem size was larger than 12, the performance of simulated annealing and genetic algorithm began to fluctuate drastically, while MIMIC still performed relatively well. Although MIMIC required the second-highest number of function evaluations and had the longest computing time, it had the highest fitness score on average and provided more stability at a larger scale than the other algorithms. Therefore MIMIC is the optimal choice for the six peaks problem.

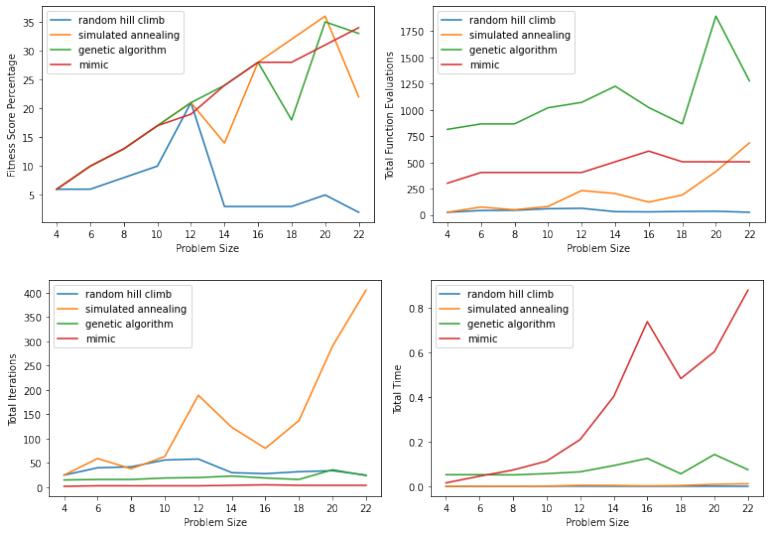


Figure 11 - Optimizer performance on different problem sizes.

4. Neural Network Optimization

Neural networks consist of multiple layers of connected neurons, which have their own weights. They typically use gradient descent and backpropagation to update their weights and biases based on the loss calculated by the loss function. This task can also be solved by randomized optimization algorithms because it is essentially an optimization problem with the loss function as its fitness function. I used a 20x28x20 network structure, the same as the previous assignment. In order to maximize the performance, I fine-tuned several hyperparameters. As shown in Figure 12, gradient descent performs better at lower learning rates and stops learning when the learning rate is too high. Randomized hill climbing and simulated annealing, on the other hand, prefer a much higher learning rate. The generic algorithm performed exactly the same at the learning rates tested. It appears to be insensitive to the change in learning rates.

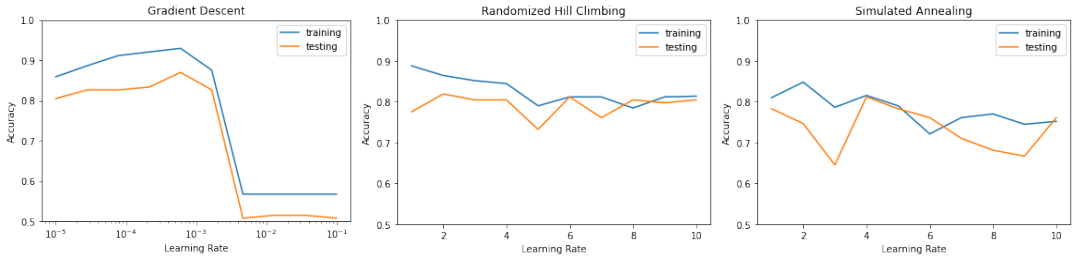


Figure 12 - Learning rate tuning.

The weight clipping parameter can limit the weights and prevent overly large values. An overly large weight can make certain features dominant in the neural network, which may lead to poor accuracy or overfitting. As shown in Figure 13, the algorithms generally have higher testing accuracy and lower overfitting with smaller clipping parameters.

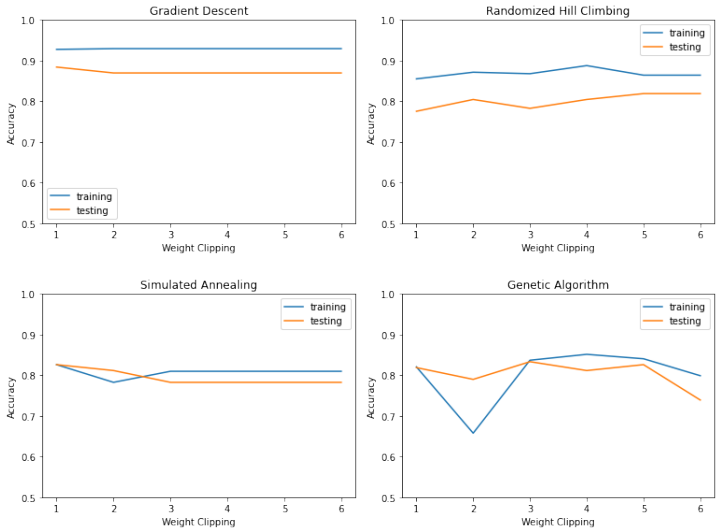


Figure 13 - Weight clipping parameter tuning.

After hyperparameter tuning, the performance of the optimized models is evaluated and shown in Figure 14 and Table 1. The two loss curve graphs focus on different regions of the graph to better highlight the characteristics. Random hill climbing had a similar lost curve to gradient descent, while simulated annealing had some noises at the beginning but started to converge after 500 iterations. The generic algorithm required much more function evaluations than the other two algorithms and therefore took significantly longer to train. The three algorithms achieved good accuracy, but none of them outperformed the gradient descent method. Random hill climbing showed signs of overfitting because its testing accuracy is noticeably lower than its training accuracy.

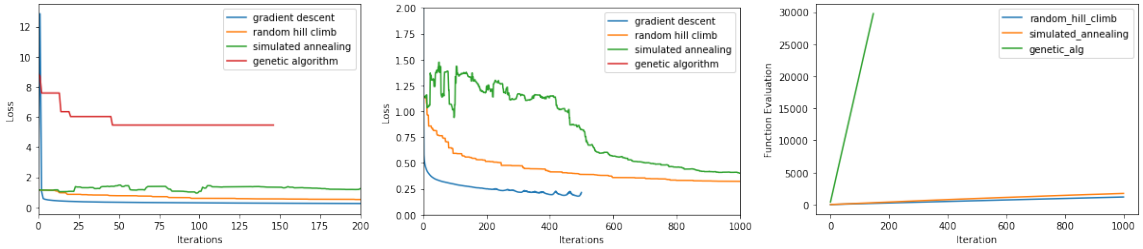


Figure 14 - Loss and function evaluation of the optimized models.

	Train Accuracy	Test Accuracy	Function Evaluation	Total Time (s)
Gradient Descent	0.928	0.884	NA	2.58
Random Hill Climbing	0.864	0.818	1178	2.15
Simulated Annealing	0.810	0.783	1657	2.80
Generic Algorithm	0.841	0.826	29754	58.88

Table 1 - Results of the optimized models.

5. Conclusion

In this project, I explored four randomized optimization algorithms in three different optimization problems. The generic algorithm outperformed the other algorithms by a small margin in the n-queen problem, while simulated annealing excelled in the flip-flop problem and was the clear winner. The algorithms performed closely in the six peaks problem, but MIMIC was the optimal choice because of its stability at larger problem sizes. In the neural network optimization experiment, the three algorithms had relatively good accuracy but did not outperform gradient descent. This is expected because gradient descent has proven to be the optimal method for many neural network architectures.

Reference

[1] Isbell, Charles L. "Randomized local search as successive estimation of probability densities." A longer tutorial version of the 1997 paper on MIMIC that includes a derivation for MIMIC with trees.