# CS7641 A2: Randomized Optimization

Scott Schmidl
*sschmidl3@gatech.edu*

## I. Introduction

The field of optimization encompasses a wide-range of algorithms and problems attempting to find the best solution to a given objective function within some number of constraints. Optimization problems pose unique challenges due to their potential for complexity large solution spaces.

In this paper, two attempts at optimization are made: 1) Solve three optimization problems: Four Peaks, Knapsack, and Flip Flop, using four different algorithms and 2) Optimize neural network weights using three different algorithms. These problems serve as benchmarks for evaluating the performance of optimization algorithms in various domains.

**Algorithms**
Random Hill Climbing (RHC) is a straight-forward random optimization algorithm used to find the maximum value of a problem. It starts with a random solution and iteratively makes random changes to the solution around a neighborhood, keeping the changes that improve the function value. Termination of the algorithm happens when it reaches a solution where no neighbor has a higher function value or it reaches a defined max attempts or max iterations. Random Hill Climbing is susceptible to getting stuck in local optima.

Simulated Annealing (SA) is a probabilistic optimization algorithm inspired by the annealing process in metallurgy that utilizes exploration and exploitation. It starts in the same way as Random Hill Climbing but employs a technique know as exploration and exploitation. This technique allows the algorithm to choose some poor solutions with the hopes that it eventually finds a path to the global optima. Simulated Annealing is more effective than Random Hill Climbing at escaping local optima and finding global optima.

Genetic Algorithm (GA) is inspired by the process of natural selection and evolution. It starts with a population of individuals, and applies selection, crossover, and mutation to generate new generations of individuals. The selection operator chooses individuals from the current individuals based on their fitness. The crossover operator combines information from two parents to create children. The mutation operator introduces small random changes to the children. Genetic Algorithm is effective at exploring large solution spaces and finding good solutions.

Mutual-Information-Maximizing Input Clustering is a probabilistic optimization algorithm that uses probability distributions to guide the search process by passing the most useful information between iterations. It starts with a random sample from a probability distribution to help capture the relationships between different variables. Then calculates a given percentile, find its distribution, and gathers new samples; the process repeats. Through this process MIMIC can pass down the best information and produce the best results.

**Optimization Problems**
The Four Peaks problem implores optimization algorithms to navigate a landscape with deceptive plateaus, where the objective is to maximize the number of leading 1's at the beginning and trailing 0's at the end of a binary string. There are two global maxima for this problem: 1) T + 1 leading 1's, where T equals the problem size times some percent, followed by all 0's; 2) T + 1 trailing 0's preceded by all 1's.

The Flip Flop problem implores optimization algorithms to maximize the number of adjacent elements in a binary string that differ. There are two global maxima for this problem: 1) the leading digit is 1 then rest alternate 0, then 1, then 0, n times; 2) the leading digit is a 0 then alternate 1, then 0, then 1, n times.

Lastly, the Knapsack problem, is a classical optimization dilemma of selecting items, that each have a weight and value, such that the value of items in the knapsack are maximized while respecting the weight capacity of the knapsack.

**Neural Network Weight Optimization**
Optimizing the weights of a neural network is a fundamental task with the goal of minimizing some loss function, and is critical for achieving high performance in various domains. Traditionally, Gradient Descent has been widely used for this purpose. However, I will explore the efficacy of alternative optimization algorithms.

In this paper, I explore the performance of four optimization algorithms—RHC, SA, GA, and MIMIC—in finding optimal weights of a neural network. Through experimentation and analysis, I strive to gain insights into the strengths and weaknesses of these algorithms in tackling different optimization problems.

I compare their results against those obtained from Gradient Descent. By evaluating the time, final performance, and robustness of these algorithms on neural network weight optimization tasks, I aim to provide insights into their suitability for this application.

## II. Experiments

### Optimization Problems

My investigation focused on maximizing three classic optimization problems: the Four Peaks problem, the Flip Flop problem, and the Knapsack problem, using four different optimization algorithms: Random Hill Climbing, Simulated Annealing, Genetic Algorithm, and Mutual-Information-Maximizing Input Clustering. Through extensive experimentation and analysis, I aimed to evaluate the efficiency of these algorithms in maximizing each objective function. Results obtained from my experiments were obtained by looking at various problem sizes. For Four Peaks and Flip Flop problem sizes 25, 50, and 75 were used, while for Knapsack problem sizes 30, 60, and 90 were used. Experiments were also performed on various parameters of the algorithms, ie restarts (RHC), temperature (SA), population size and mutation rates (GA), and population size and keep percent (MIMIC). The experimentation revealed distinct performance characteristics of each optimization algorithm across the different optimization problems.

For the Four Peaks problem, RHC demonstrated good efficiency in consistently climbing the hill to a maxima. SA offered a balance between exploration and exploitation with clear signs of reducing the fitness in order to hopefully reach the global optima. GA shows signs of an ability to reach the global optima in few iterations. MIMIC was similar to GA in that it found an optima quite quickly.

In the case of the Flip Flop problem, similar to Four Peaks, Random Hill Climbing demonstrated good efficiency in consistently climbing the hill to a maxima. Simulated Annealing became more erratic with it's exploration and exploitation technique as temperature increased. Genetic Algorithm also performed well, finding optima with few iterations required. MIMIC was similar to GA in that it found an optima quite quickly.

For the Knapsack problem, Random Hill Climbing struggling to find global optima in several case was did have success. Again, Simulated Annealing become more erratic with increases in temperature but was able to find the global optima. Genetic Algorithm had great success in finding global optima by selecting a subset of items that maximized the total value while respecting the weight capacity constraint. MIMIC, like GA, had great success in find the optima.

### Neural Network Weight Optimization

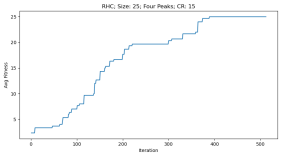
My experimentation involved optimizing the weights of a neural network using mlrose-hiive and employing RHC, SA, GA, and Gradient Descent. I conducted a comprehensive comparative analysis of these algorithms to evaluate their performance in terms of time and performance. Gradient Descent from scikit-learn was compared to Gradient Descent from mlrose-hiive, and both of those compared to RHC, SA, and GA from mlrose-hiive. While Gradient Descent performed better than RHC and SA, GA showed a lot of promise for future implementations, when based solely on performance. However, the fitting time for GA needs to be weighted into any decision making process as GA took more than 10 times longer than any other algorithm, on average.

## III. Analysis

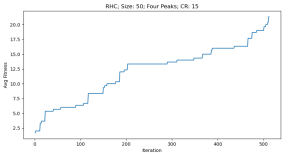
### Optimization Problems

#### Four Peaks

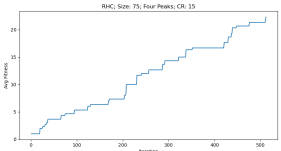
In figure 1, I have examined RHC for a specific restart and three different state sizes. In figure 1a, I review a state size of 25, figure 1b, a state size of 50, and figure 1c a state size of 75. After 512 iterations, the global optima was not reach. In this case, RHC got a stuck in a local optima.



(a) Current Restart 15/15, State Size 25



(b) Current Restart 15/15, State Size 50



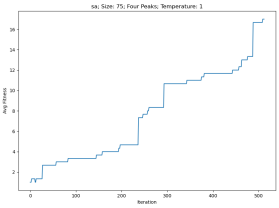
(c) Current Restart 15/15, State Size 75

Fig. 1: Four Peaks, RHC, Different Sizes

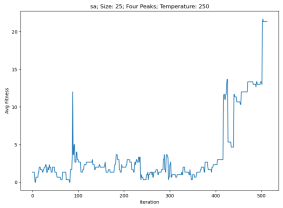
In figure 2, I have examined SA for four different temperatures and three different state sizes. In figure 2a, I review a temperature of 1 with state size of 75, figure 2b, a temperature of 250 with a state size of 25, in figure 2c, a temperature of 250 with state size of 50, and in figure 2d a temperature of 25 with a state size of 75. After 512 iterations, it is easy to see that SA spends a significant amount of time walking down the hill with the hopes of finding the correct path to the peak. This is the exploration and exploitation technique. It appears that SA struggled here to find the global optima.

In figure 3, I have examined GA for three different population sizes, the same mutation rate, and the same state size. In figure 3a, I review a population size of 100, mutation rate of 0.6, and state size of 75. In figure 3b, I review a population size of 150, mutation rate of 0.6, and state size of 75. Finally, in figure 3c, I review a population size of 200, mutation rate of 0.6, and state size of 75. GA does a great job of finding a good fitness in under 200 iterations.

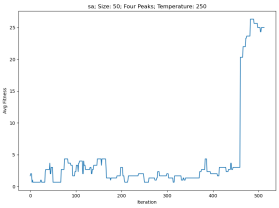
In figure 4, I have examined MIMIC for the same keep percent and population size, but different state sizes. In



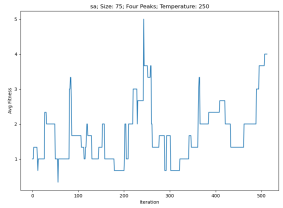
(a) Tempearture 1, State Size 75



(b) Tempearture 250, State Size 25

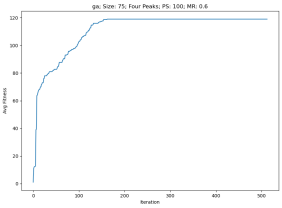


(c) Tempearture 250, State Size 50

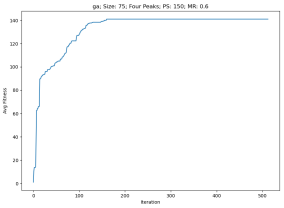


(d) Tempearture 250, State Size 75

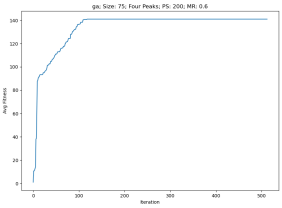
Fig. 2: Four Peaks, SA, Different Sizes



(a) Population Size 100, Mutation Rate 0.6, State Size 75



(b) Population Size 150, Mutation Rate 0.6, State Size 75



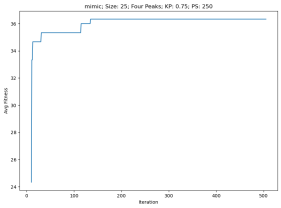
(c) Population Size 200, Mutation Rate 0.6, State Size 75

Fig. 3: Four Peaks, GA, Different Sizes

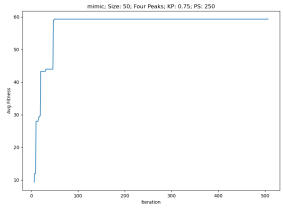
figure 4a, I review a keep percent of 0.75, a population size of 250, and state size of 25. In figure 4b, I review a keep percent of 0.75, a population size of 250, and state size of 50. In figure 4c, I review a keep percent of 0.75, population size of 250, and state size of 75. MIMIC does a great job of getting to an optimum solution in under 150 iterations. Due to the nature of MIMIC, which uses probability distributions to sample data, this model has great potential for problems such as Four Peaks.

**Flip Flop**

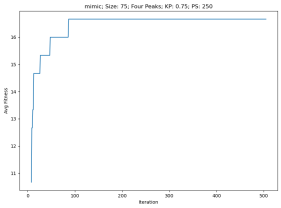
In figure 5, I have examined RHC for a specific restart and three different state sizes. In figure 5a, I review a



(a) Keep Percent 0.75, Population Size 250, State Size 25



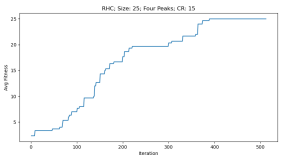
(b) Keep Percent 0.75, Population Size 250, State Size 50



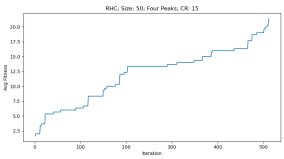
(c) Keep Percent 0.75, Population Size 250, State Size 75

Fig. 4: Four Peaks, MIMIC, Different Sizes

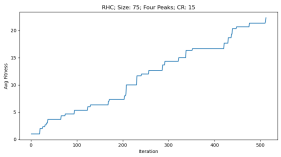
state size of 25, figure 5b, a state size of 50, and figure 5c a state size of 75. After 512 iterations, the global optima was not reach.



(a) Current Restart 15/15, State Size 25



(b) Current Restart 15/15, State Size 50

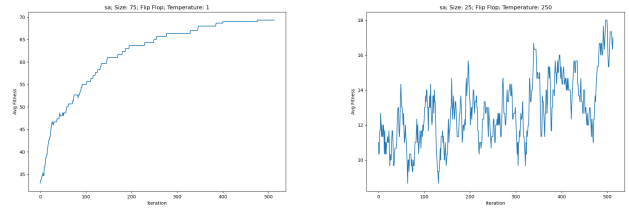


(c) Current Restart 15/15, State Size 75

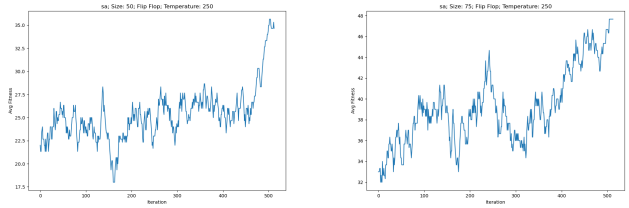
Fig. 5: Flip Flop, RHC, Different Sizes

In figure 6, I have examined SA for four different temperatures and three different state sizes. In figure 6a, I review a temperature of 1 with state size of 75, figure 6b, a temperature of 250 with a state size of 25, in figure 6c, a temperature of 250 with state size of 50, and in figure 2d a temperature of 25 with a state size of 75. After 512 iterations, it is easy to see that SA spends a significant amount of time walking down the hill with the hopes of finding the correct path to the peak. This is the exploration and exploitation technique. It appears that with a temperature of one, SA was able to find the correct peak and reached a global optima. However, with a temperature of 250, not only did SA struggle to find a

global optima, but it spent a significant amount of time moving down hill resulting in erratic behavior.



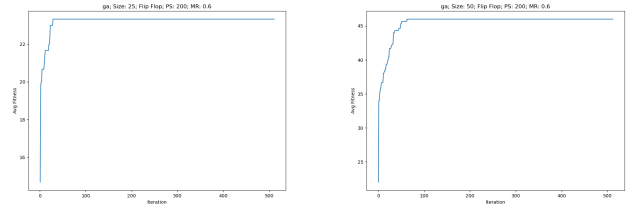
(a) Temperature 1, State Size 75 (b) Temperature 250, State Size 25



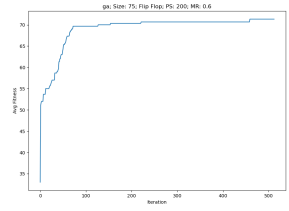
(c) Temperature 250, State Size 50 (d) Temperature 250, State Size 75

Fig. 6: Flip Flop, SA, Different Sizes

In figure 7, I have examined GA for the same population size and the same mutation rate, but different state sizes. In figure 7a, I review a population size of 200, mutation rate of 0.6, and state size of 75. In figure 7b, I review a population size of 200, mutation rate of 0.6, and state size of 50. Finally, in figure 7c, I review a population size of 200, mutation rate of 0.6, and state size of 75. GA, again, does a great job of finding an optima in under 100 iterations.



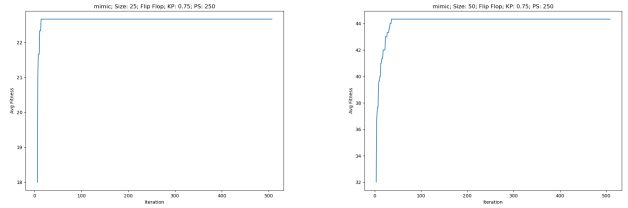
(a) Population Size 200, Mutation Rate 0.6, State Size 25 (b) Population Size 200, Mutation Rate 0.6, State Size 50



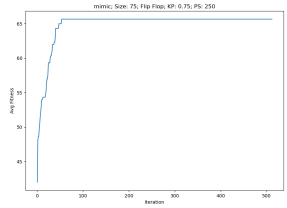
(c) Population Size 200, Mutation Rate 0.6, State Size 75

Fig. 7: Flip Flop, GA, Different Sizes

In figure 8, I have examined MIMIC for the same keep percent and population size, but different state sizes. In figure 8a, I review a keep percent of 0.75, a population size of 250, and state size of 25. In figure 8b, I review a keep percent of 0.75, a population size of 250, and state size of 50. In figure 4c, I review a keep percent of 0.75, population size of 250, and state size of 75. MIMIC does a great job of getting to an optimum solution in under 100 iterations. Due to the nature of MIMIC, it can perform quite well with problem similar to Flip Flop



(a) Keep Percent 0.75, Population Size 250, State Size 25 (b) Keep Percent 0.75, Population Size 250, State Size 50

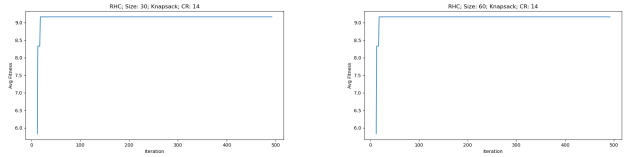


(c) Keep Percent 0.75, Population Size 250, State Size 75

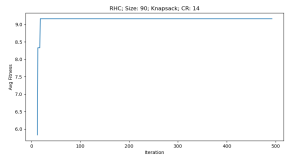
Fig. 8: Flip Flop, MIMIC, Different Sizes

**Knapsack**

In figure 9, I have examined RHC for a specific restart and three different state sizes. In figure 9a, I review a max item count of 30, figure 9b, a max item count of 60, and figure 9c a max item count of 90. After 512 iterations, the global optima was not reach. In this case, RHC struggled to make improvements and got stuck in a local optima.



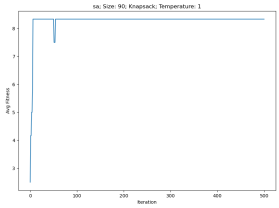
(a) Current Restart 14/15, State Size 30 (b) Current Restart 14/15, State Size 60



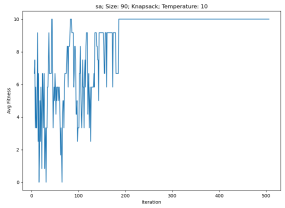
(c) Current Restart 14/15, State Size 90

Fig. 9: Knapsack, RHC, Different Sizes

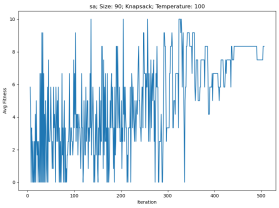
In figure 10, I have examined SA for four different temperatures and one max item count. In figure 10a, I review a temperature of 1 with max item count of 90, figure 10b, a temperature of 250 with a max item count of 90, in figure 10c, a temperature of 250 with max item count of 90, and in figure 10d a temperature of 25 with a max item count of 90. After 512 iterations, it is easy to see that in figure 10a, SA gets stuck in a local optima. Furthermore, in figures 10b, 10c, and 10d I observe erratic behavior from SA, in that, it spends a significant amount of time walking down the hill with the hopes of finding the correct path to the peak. All that to say it appears that SA struggled here to find the global optima, but it did eventually find it.



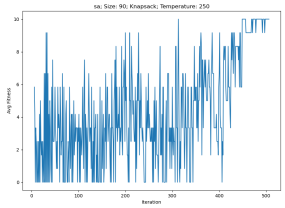

(a) Temperature 1, State Size 90




(b) Temperature 10, State Size 90




(c) Temperature 100, State Size 90

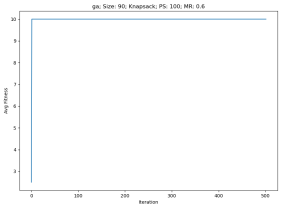



(d) Temperature 250, State Size 90

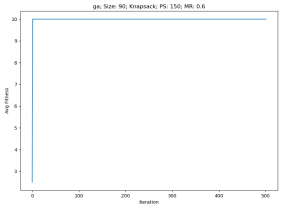
Fig. 10: Knapsack, SA, Different Sizes

In figure 11, I have examined GA for three different population sizes, the same mutation rate, and the same state size. In figure 11a, I review a population size of 100, mutation rate of 0.6, and state size of 90. In figure 11b, I review a population size of 150, mutation rate of 0.6, and state size of 90. Finally, in figure 11c, I review a population size of 200, mutation rate of 0.6, and state size of 90. GA does a great job of finding a good fitness score and an optima quite rapidly, in this case. It's almost instantaneous, and investigation should be done into these results.

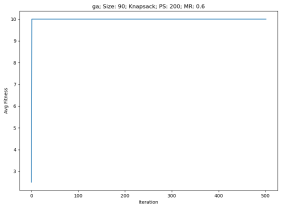
In figure 12, I have examined MIMIC for the same keep percent and population size, but different state sizes. In figure 12a, I review a keep percent of 0.75, a population size of 250, and state size of 30. In figure 12b, I review a keep percent of 0.75, a population size of 250, and state size of 60. In figure 12c, I review a keep percent of 0.75, population size of 250, and state size of 90. For the Knapsack problem, MIMIC gets to an optimal solution instantaneously. This could largely be due to the parameters set for Knapsack allowing for




(a) Population Size 100, Mutation Rate 0.6 State Size 90



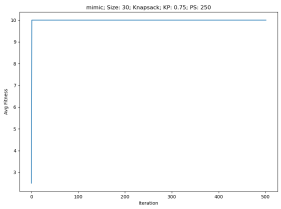

(b) Population Size 150 , Mutation Rate 0.6 State Size 90



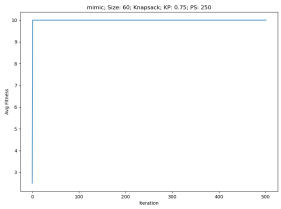

(c) Population Size 200, Mutation Rate 0.6 State Size 90

Fig. 11: Knapsack, GA, Different Sizes

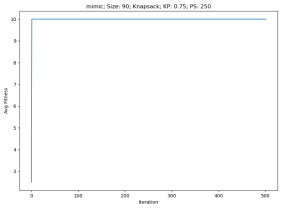
MIMIC to resolve it simply and quickly.




(a) Keep Percent 0.75, Population Size 250, State Size 30




(b) Keep Percent 0.75, Population Size 250, State Size 60




(c) Keep Percent 0.75, Population Size 250, State Size 90

Fig. 12: Knapsack, MIMIC, Different Sizes

**Neural Network Weight Optimization**

In TABLE 1, the results obtained from scikit-learn are shown. It exhibited decent convergence relative to the other algorithms converging in 9.2 seconds and achieved competitive performance with a training recall of 63.7% and validation recall of 63.7%. There doesn't appear to be any overfitting, underfitting, nor data leakage for this model.

In TABLE 2, the results obtained from RHC via MLROSE are shown. It exhibited poor convergence relative

TABLE I: Gradient Descent (scikit-learn: MLP)

| Fit Time (Seconds) | Recall Train | Recall Validation |
|---|---|---|
| 9.2 | 63.7 | 63.7 |

to gradient descent converging in over 30 seconds for each combination and achieved worse performance than scikit-learn and GA with a training recall ranging from 49.8% to 56.7% and validation recall ranging from 57.1% to 59.3%. There doesn't appear to be any overfitting but there does appears to be some data leakage as the validation was higher than training. This possible data leakage should be investigated.

TABLE II: Random Hill Climbing

| Fit Time (Seconds) | Learning Rate | Restarts | Recall Train | Recall Validation |
|---|---|---|---|---|
| 32.2 | 0.01 | 5 | 54.6% | 57.1% |
| 64.7 | 0.05 | 10 | 56.7% | 59.3% |
| 95.3 | 0.10 | 15 | 49.8% | 57.1% |

In TABLE 3, the results obtained from SA via MLROSE are shown. It exhibited the best convergence relative to gradient descent and the other algorithms converging in under 8 seconds and achieved slightly better performance than RHC and GD (mlrose) but slightly worse than scikit-learn with a training recall ranging from 54.2% to 59.7% and validation recall ranging from 56.5% to 63.0%. There doesn't appear to be any overfitting but there does appear to be some data leakage as the validation was higher than training. This possible data leakage should be investigated.

TABLE III: Simulated Annealing

| Fit Time (Seconds) | Learning Rate | Decay | Recall Train | Recall Validation |
|---|---|---|---|---|
| 7.5 | 0.01 | Geom | 54.2% | 56.5% |
| 6.8 | 0.05 | Arith | 57.5% | 59.4% |
| 6.3 | 0.10 | Exp | 59.7% | 63.0% |

In TABLE 4, the results obtained from GA via MLROSE are shown. It exhibited the worst convergence relative to gradient descent and the other algorithms converging in over 650 seconds but achieved the best performance with a training recall ranging from 67.4% to68.0% and validation recall ranging from 64.4% of 67.8%. There doesn't appear to be overfitting, underfitting, nor data leakage. The lack of evidence for data leakage in this model could be evidence proving that the other algorithms, with apparent leakage, randomly performed better validation.

TABLE IV: Genetic Algorithm

| Fit Time (Seconds) | Learning Rate | Population Size | Mutation Rate | Recall Train | Recall Validation |
|---|---|---|---|---|---|
| 695.3 | 0.01 | 175 | 0.05 | 67.4% | 66.8% |
| 803.3 | 0.05 | 200 | 0.10 | 68.0% | 67.8% |
| 888.7 | 0.10 | 225 | 0.15 | 65.3% | 64.4% |

In TABLE 5, the results obtained from GD via MLROSE are shown. It exhibited decent convergence relative to scikit-learn and the other algorithms converging in under 9.75 seconds. However, this model achieved moderate performance that is slightly worse than scikit-learn and comparable to RHC and SA. It achieved a training recall of 54.0% and validation recall of 56.0%.

TABLE V: Gradient Descent (MLROSE)

| Fit Time (Seconds) | Recall Train | Recall Validation |
|---|---|---|
| 9.75 | 54.0 | 56.0 |

## IV. Conclusion

In conclusion, my study sheds light on the effectiveness of various optimization algorithms in addressing classic optimization problems. My findings indicate that each algorithm exhibits distinct characteristics and performs differently across different domains. While RHC demonstrates simplicity and efficiency, SA offers a balance between exploration and exploitation, GA excels in exploring solution spaces and escaping local optima and, finally, MIMIC showcases the power of probabilistic models in guiding the search process.

Through empirical analysis, I have observed the behavior of RHC, SA, GA, and MIMIC on the Four Peaks problem, the Knapsack problem, and the Flip Flop problem. My study highlights the importance of selecting appropriate optimization algorithms based on the characteristics of the optimization problem. By understanding the strengths and weaknesses of different algorithms, we can make informed decisions in designing and implementing optimization solutions for real-world problems.

Furthermore, my study provides valuable insights into the performance of RHC, SA, and GA in optimizing the weights of neural networks, compared to Gradient Descent. My results demonstrate that while Gradient Descent remains a powerful optimization technique for neural network weight optimization, RHC, SA, and GA also exhibit promising performance.

Finally, these algorithms offer different trade-offs in terms of time, space, performance, and robustness, mak-

ing them valuable techniques for optimization. Moreover, my findings suggest that the choice of optimization algorithm may depend on various factors, including the specific characteristics of the problem architecture, the nature of the optimization problem, and computational resources available. By understanding the strengths and weaknesses of different optimization algorithms, practitioners can make informed decisions in selecting the most suitable approach for their optimization tasks, ultimately leading to improved model performance and efficiency.

## V. Resources


[1] *API Reference.* Python. https://www.python.org/.
[2] *API Reference.* Pandas. https://pandas.pydata.org/.
[3] *API Reference.* Jupyter. https://jupyter.org/.
[4] *API Reference.* IPython. https://ipython.org/.
[5] *API Reference.* MatPlotLib. https://matplotlib.org/stable/.
[6] *API Reference.* Scikit-Learn. https://scikit-learn.org.
[7] Nakamura, K. (2023). *ML LaTeX Template*.
[8] Rollings, A. 2020. *mlrose: Machine Learning, Randomized Optimization and SEarch package for Python, hiive extended remix* https://github.com/hiive/mlrose