

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 known 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 probabilistic 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, finds 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 RHC SA GA MIMIC
Flip Flop RHC SA GA MIMIC
Knapsack RHC SA GA MIMIC

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.

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

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

In TABLE 2, the results obtained from RHC via ML-ROSE are shown. It exhibited poor convergence relative 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
32.2	0.01	5	54.6%
64.7	0.05	10	56.7%
95.3	0.10	15	49.8%
Recall Validation			
57.1%			
59.3%			
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
7.5	0.01	Geom	54.2%
6.8	0.05	Arith	57.5%
6.3	0.10	Exp	59.7%
Recall Validation			
56.5%			
59.4%			
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% to 68.0% and validation recall ranging from 64.4% to 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
695.3	0.01	175
803.3	0.05	200
888.7	0.10	225
Mutation Rate		
0.05	Recall Train	Recall Validation
0.10	67.4%	66.8%
0.15	68.0%	67.8%
	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, making 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>