José Tristani | jtristani3

Charles Isbell

CS-7641: Machine Learning

October 17, 2021

Randomized Optimization Analysis

This analysis will evaluate and contrast the different methods in Randomized Optimization, the methods in question are: Randomized Hill Climbing, Simulated Annealing, Genetic Algorithms and MIMIC. This analysis will also use the first three of these methods to train the weights of our last analysis's DNN implementation for the Poker Dataset. The use of this dataset will be interesting with these random search algorithms given that the original implementation using gradient descent achieved remarkable performance, it was able to infer that hand position is irrelevant and thus achieve almost perfect performance in the testing dataset, this peculiarity is important given the nature of the problem and the dataset.

First, we will contrast the different methods across 3 different problem domains to and discuss their behavior, strengths, and weaknesses. The problem domains will be N-Queens, Six Peaks, and FlipFlop.

Starting with a very well-known problem, the N-Queens problem. The problem is such that if there were a chess board with 8 queens, the fitness function will evaluate if the queens are arranged in such a way that no two queens can attack each other. As shown in Figure 1, which is composed of the different fitness function results for the 4 methods being tested. The highest score achieved belonging to the Genetic Algorithm method is approximately 28, with MIMIC and Randomized Hill Climbing achieving similar results, and Simulated Annealing has oscillated into various local maxima, but did not achieve a higher fitness score than the other methods.

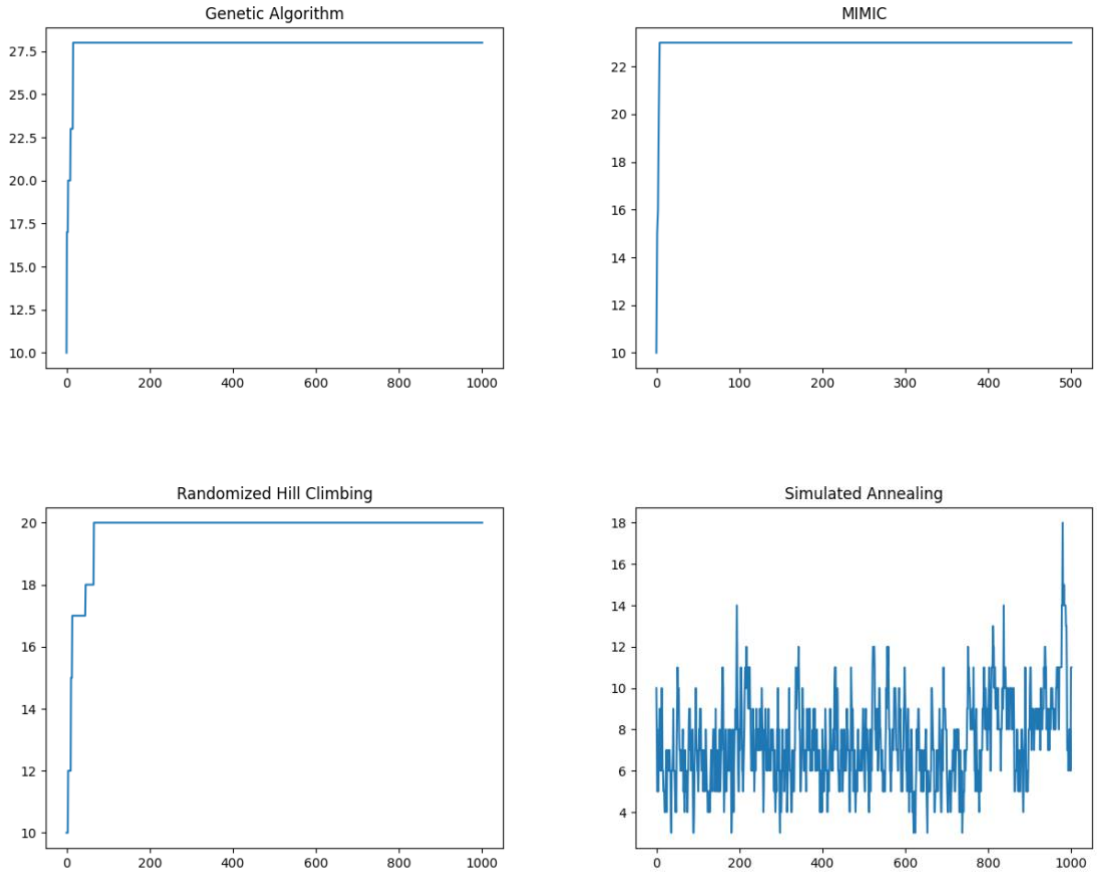

Figure 1: N-Queens problem

Next is the Six Peaks problem, and like the N-Queens problem, the best performer here is also Genetic Algorithm followed by MIMIC, although this is a very tough problem, it is surprising to see MIMIC struggle along with the others. GA leverages its knowledge from generation to generation to find better maxima, like MIMIC, but it seems that MIMIC, which should generally use less iterations, reaches the upper limit and is still not at its maxima. Genetic Algorithms does vastly outperform the others in six peaks. In terms of run time, the fastest were Simulated annealing and Random hill climbing, followed by Genetic Algorithm and lastly MIMIC, which in terms of wall clock time would run in 3-5 minutes, versus the rest, which would finish within 10-30 seconds each. This drawback makes MIMIC very slow to iterate with.

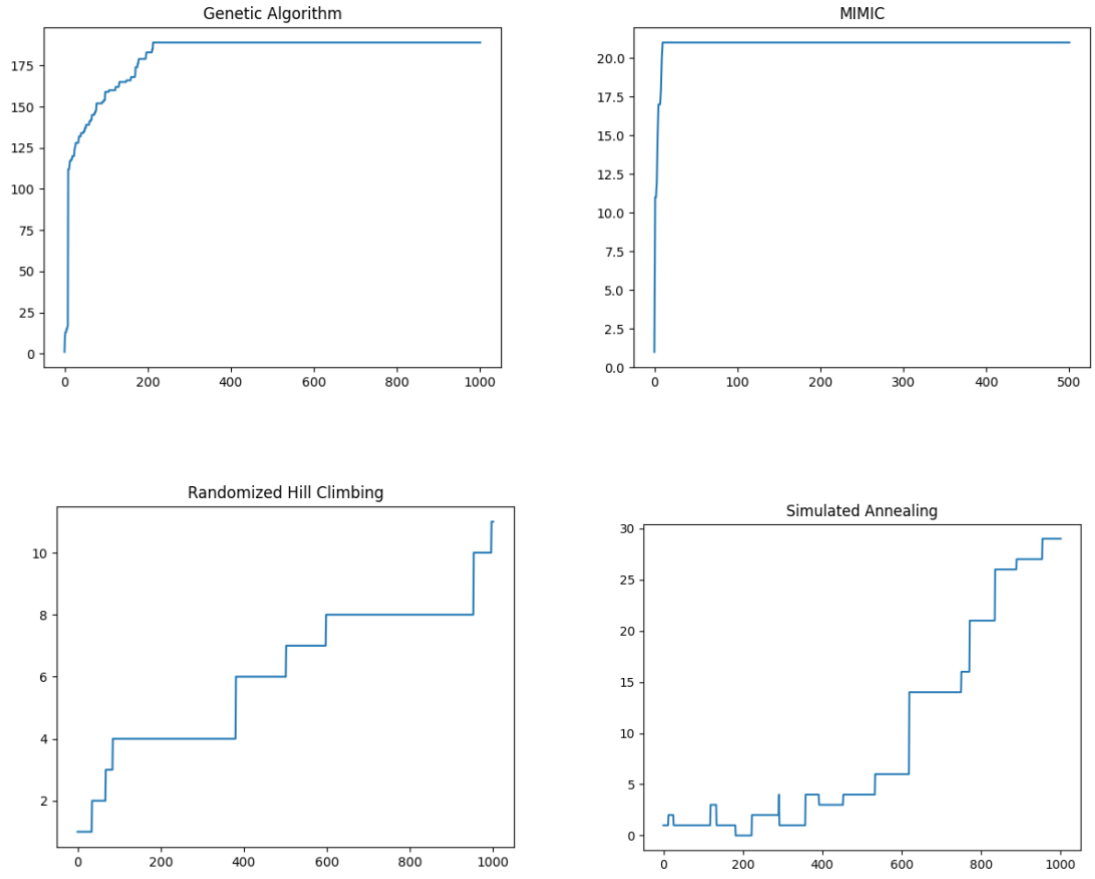

Figure 2: Six Peaks problem

Lastly, the last problem that will be used to contrast these methods is the Flip-Flop problem. The function evaluates the fitness of a vector *x* as the total number of pairs of consecutive elements of *x*. With this problem, it's harder than it seems with results, even though Genetic comes out at top again with the best overall fitness score, MIMIC achieves a similar score, albeit lower, but just within its first 10 iterations, comparable to the performance of GA, which took upwards of 700 iterations to achieve. Randomized Hill Climbing also achieves similar scores, and Simulated Annealing comes in last once again. Simulated annealing's temperature hyperparameter, which gives it its distinct advantage seems to be holding it back in these domains as it has struggled to

outperform the other methods, these results were achieved using the Geometric Decay for the temperature of Simulated annealing.

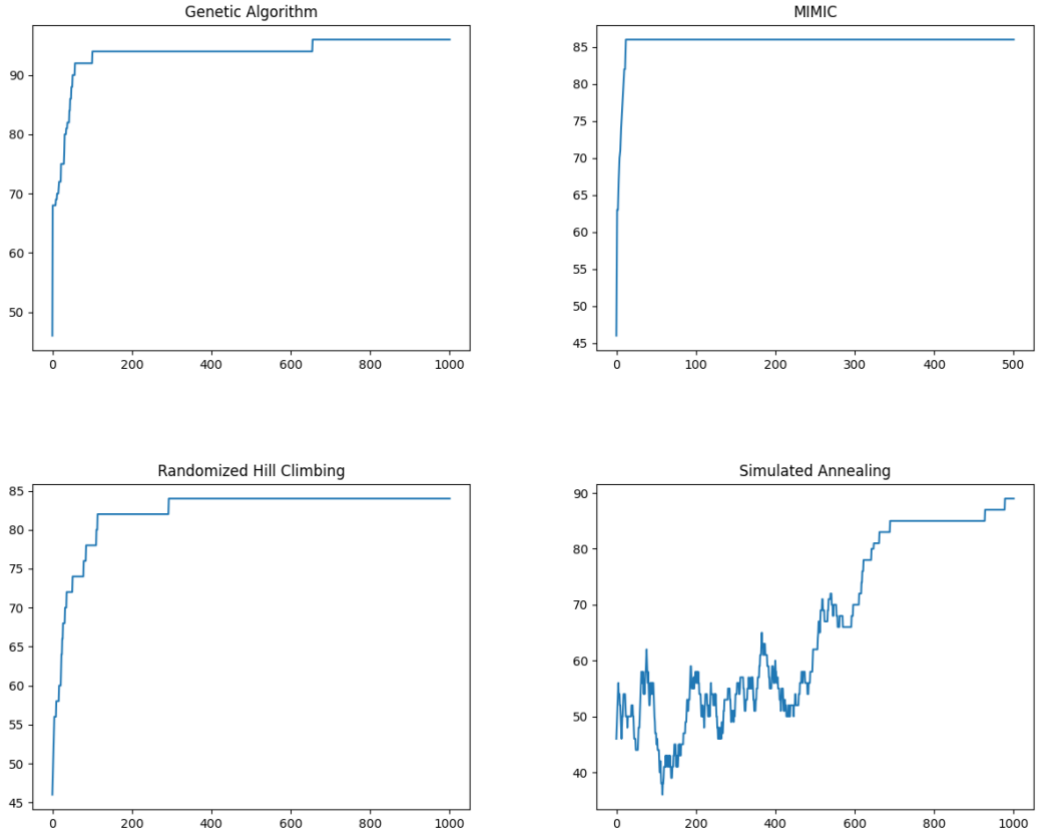

Figure 3: Flip-Flop problem results

Lastly, Randomized Hill Climbing, Simulated Annealing, and Genetic Algorithms were used to train the weights of the Deep Neural Network that was used in the past report on Supervised Learning algorithms. With little surprise, none of the methods achieved a better performance in our tests. The results were very similar across the board, and the wall clock times were unrealistically high, making using them for finding optimal weights time consuming and impractical. Back propagation, initially achieves incredible results in the poker dataset, making it very hard to beat, much less get equal results. With backpropagation, the DNN achieves above

99% accuracy without much effort and time. This is in part since the poker dataset has been specifically crafted so that unlikely scenarios are oversampled in the training data. This does not help with randomized optimization given the fact that the dataset's testing data is realistic and is mostly comprised of no-hands, percentage wise they amount to about a 50% of the training data. This followed by the fact that some of the hands are very improbable leads to a large number of local minima and a needle in a haystack scenario, with which most of these randomized optimization methods will struggle. There is very little to optimize with regards to features in this dataset, and dimensionality is small as is. With very little success the MADELON dataset, the other dataset from the past analysis resulted in the same results, worse accuracy and impractical training times, dimensionality reduction will most definitely improve results with the MADELON dataset.

Algorithm	*Train set Accuracy*	*Test set Accuracy*	*Time*
Randomized Hill Climbing	49.95%	50.10%	8 mins
Simulated Annealing	49.94%	50.11%	10 mins
Genetic Algorithms	~48.50%	~48.00%	~45 mins
Backpropagation	99.99%	98.82%	~1 min

In conclusion, the best performer overall across both the domain problems as well as the DNN weights was Genetic Algorithms followed by Randomized Hill Climbing. Genetic Algorithms are unique in this area in that they carry memory from generation from generation, as well as the random mutations are there to help the population escape from local maxima scenarios, this is evident in the results. And as the results mention, MIMIC is close second, but its run time is the slowest of the bunch, and by a few orders of magnitude too. This means that Randomized Hill Climbing, given the time that MIMIC runs, makes for a better choice. And although these results have their pros/cons, not one of the methods is a good candidate to train neural network weights in neither the Poker nor MADELON dataset. The results are very close

to the percentage of no-hand card combinations, giving little to no value to the classification power, thus by this percentage it can be concluded that random optimization methods yielded no valuable results for the poker dataset. Given more time, a good experiment would be to use an impractical amount of time (Days? Weeks?) to see if one of these methods achieve similar performance to Backpropagation.

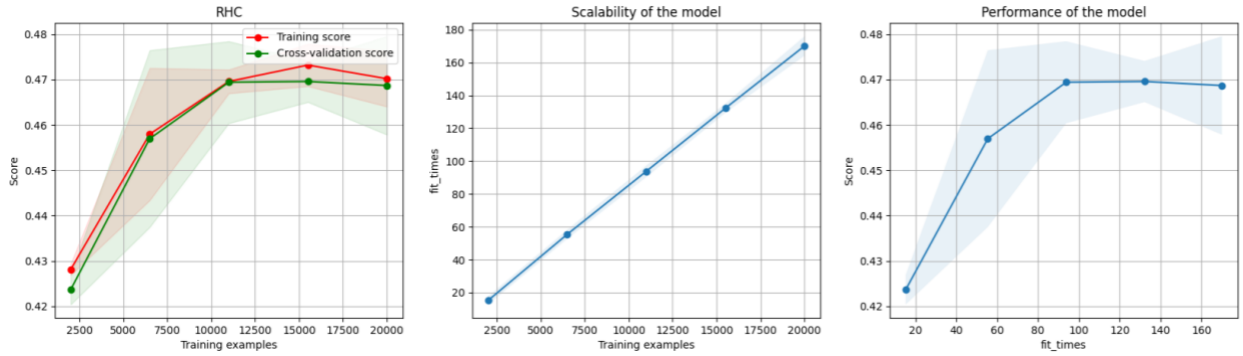

Figure 4: Randomized Hill Climbing learning curve, using Poker Dataset

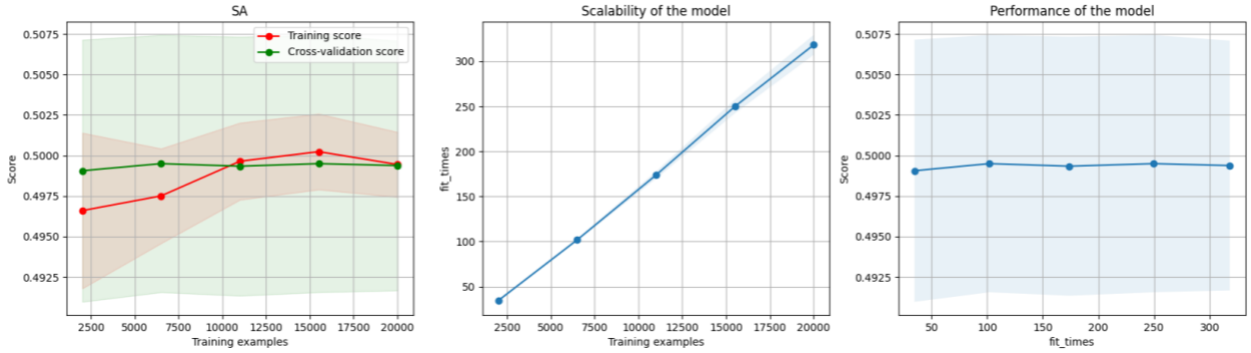

Figure 5: Simulated Annealing DNN Learning Curve, with Poker dataset

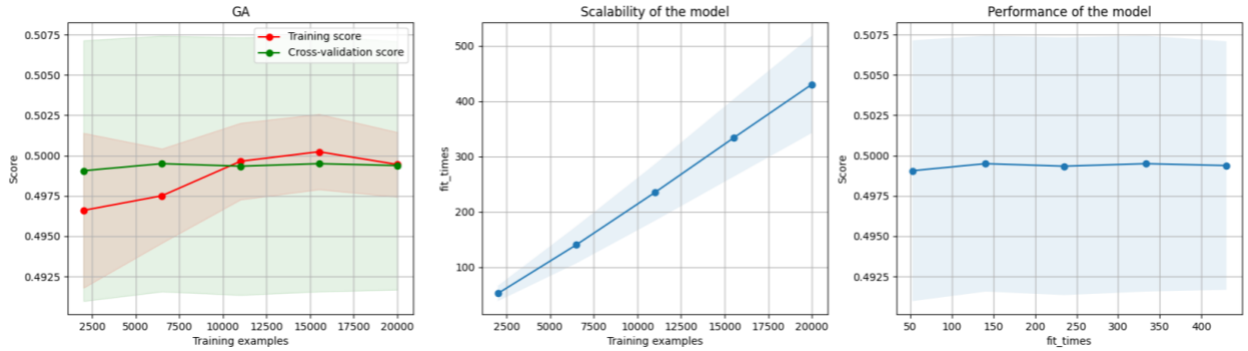

Figure 6: Genetic Algorithm DNN Learning Curve, with Poker Dataset

Works Cited (A.)

A., Rollings. n.d. *Machine LEarning, Randomized Optimization and Search Package for Python, hiive extended Remix.* <https://github.com/hiive/mlrose>.

Cattral, Robert and Oppacher Franz. "Poker Hand Data Set ." 1 January 2007. *UCI Machine Learning Repository [https://archive.ics.uci.edu/ml/datasets/Poker+Hand].* 20 September 2021.

developers, Scikit-learn. "Plotting Learning Curves." 2021. *Scikit Learn.* 16 September 2021. <https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html>.

Dua, D and C Graff. *UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]*. 2019. 15 September 2021.

G, Hayes. "MLRose." n.d. *Github.* October 2021. <https://github.com/gkhayes/mlrose>.

Guyon, Isabelle, et al. *UCI: Machine Learning Repository [https://archive.ics.uci.edu/ml/datasets/madelon]*. 2004. 18 September 2021.