Yi Shuen Lim
ylim68 | 903647878
CS 7641: Machine Learning
Assignment 2

Randomized Optimization

Introduction

This report explores the performance of different randomized optimization algorithms in two main sections. The optimization algorithms tested are: Randomized Hill Climbing (RHC), Simulated Annealing (SA), Genetic Algorithms (GA) and the Mutual Information Maximization for Input Clustering (MIMIC). First, the relative performance will be evaluated on three discrete optimization problems, namely, the Traveling Salesman Problem, the Continuous Peaks Problem, and the Flip Flop Problem. This first section was implemented using the *mlrose-hiive* library in Python. In the second section, RHC, SA and GA will be tested against backpropagation for optimizing a neural network for detecting credit card fraud. This second section was implemented using *ABAGAIL* in Jython.

The Optimization Algorithms

Randomized Hill Climbing (RHC): Optimal values are found by randomly starting at a position, and then moving toward neighboring positions that yield better performance (fitness). Once maximum fitness is achieved, and this will be a local optimum, another position is randomly selected and the process repeated. The position that yields the highest fitness across multiple "hill climbs" will be selected as the optimal value. RHC is not computationally complex, but if a problem is complex and has a lot of peaks or local optima, it can take a long time to converge.

Simulated Annealing (SA): SA helps mitigate the issue of slow convergence in RHC by implementing a probability function to decide where over the space to select its next search starting position. This function evaluates the fitness difference between the current and new position, divides it by a temperature value that decays per iteration. As temperature decreases, the probability of finding a better peak increases. This allows for a more probabilistic approach to finding global optima. Though probabilistically better at finding optimal values compared to RHC, SA can take a long time if the problem is complex.

Genetic Algorithm (GA): From an initial set of randomly generated solutions, better solutions are sampled based on their fitness and new offspring solutions are generated. They are generated via crossover -- where parts of these parent solutions are spliced with each other, as well as through mutation -- where random values of a solution are flipped. GA works especially well when the fitness function does not have a derivative and provides a more holistic random way to search the population of solutions compared to RHC and SA. However, when the optimal solution isn't dependent on the location of bits within an encoded bit string, it might not be able to converge.

Mutual Information Maximization for Input Clustering (MIMIC): Optimal values are found using joint probability density functions of input features, modeling them relative to fitness. These joint probabilities of input features are visually represented through dependency trees, and a hyperparameter that controls how much dependency the child nodes have on the parent node can be tuned. Solutions are sampled iteratively based on these fitness values until an optimal solution that has an absolute probability of 1 is found. This method, though effective, is its relatively long runtime.

Part 1: Discrete Optimization Problems

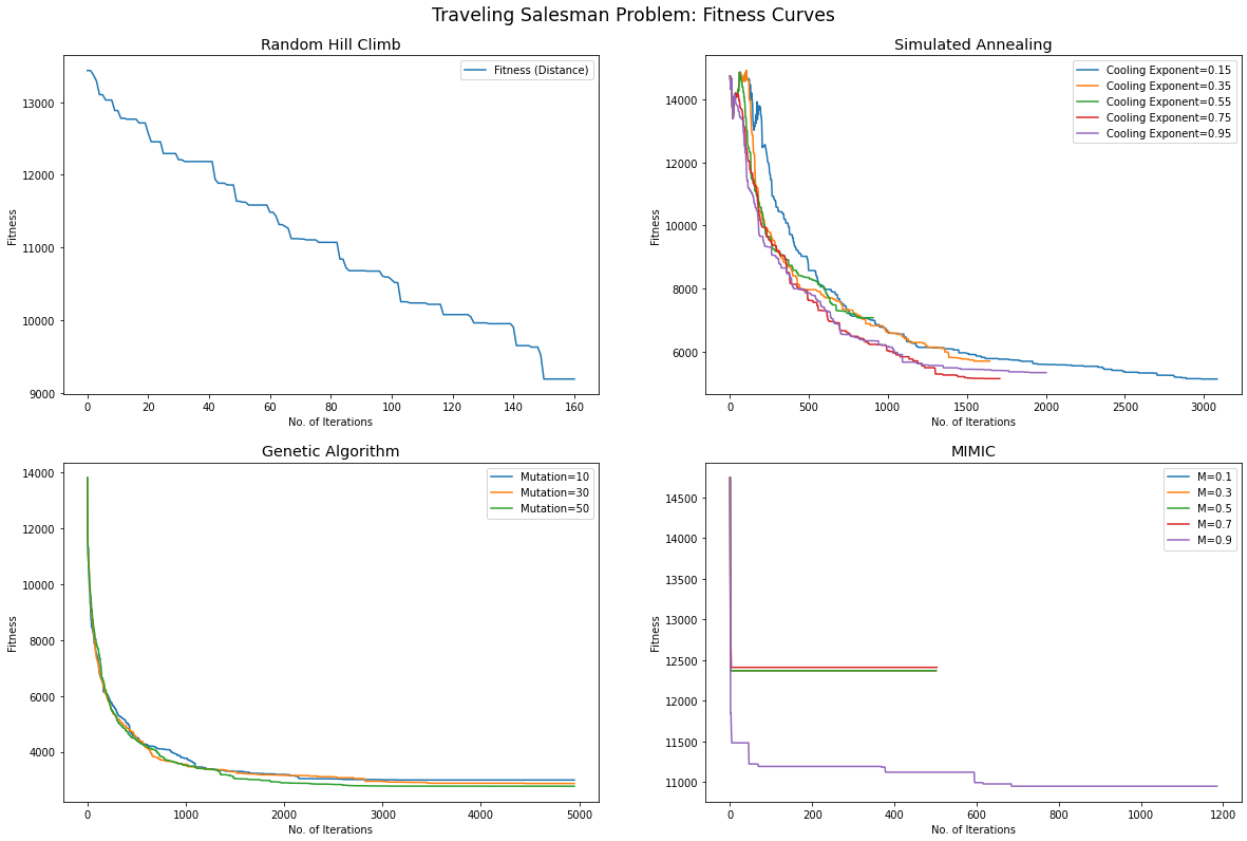
For each of the discrete optimization problems, the four aforementioned optimization techniques were tested for up to 5000 iterations with the following hyperparameters:

- Randomized Hill Climbing: No hyperparameters
- Simulated Annealing:
    - Cooling exponents [0.15, 0.35, 0.55, 0.75, 0.95]
- Genetic Algorithm: Population = 100, Crossover = 0.5
    - Mutation rates [0.1, 0.3, 0.5]
- MIMIC: Population = 100
    - M [0.1, 0.3, 0.5, 0.7, 0.9]

For each discrete optimization problem, we'll see how the value of the fitness function associated with each problem changes over the iterations to evaluate the performance of the different optimization algorithms. We'll also take a look at the time each optimization technique took to run the maximum number of iterations.

Traveling Salesman Problem

Given a number of cities and distances between each pair of cities, in this case we tested for 100 cities, the traveling salesman problem (TSP) aims to find the shortest route that visits every city and returns to the starting city. TSP's fitness is a minimization problem, so we're aiming to find the lowest value of fitness. Below are the results of the four optimization algorithms on this problem.



Looking at the y-axis of each subplot, we see that GA with a mutation parameter set at 30 outperforms the other optimization techniques substantially, achieving a final fitness score roughly half of the best SA run and less than a third of RHC. MIMIC performed the poorest for this problem, with M=0.9 being the best iteration, but still with a fitness score about four times that of GA.

Randomized Optimization	Best Fitness	# Iter to Optima	Average Time (seconds)
Randomized Hill Climbing	9105.638	186	0.320
Simulated Annealing	5130.209 (CE=0.15)	2699	0.368
Genetic Algorithm	2588.496 (Mutation=30)	4984	32.879
MIMIC	10949.147 (M = 0.9)	685	1055.972

The solution space of TSP consists of many local optima, as there could be many routes that have similar fitness values but have significantly different paths, but only one global optimal route. GA is able to look for the commonalities in well-performing solutions, narrowing down sub-routes such as clusters of cities, and work on optimizing the path around those low-distance clusters thanks to its structure of doing crossovers across different low-distance routes.

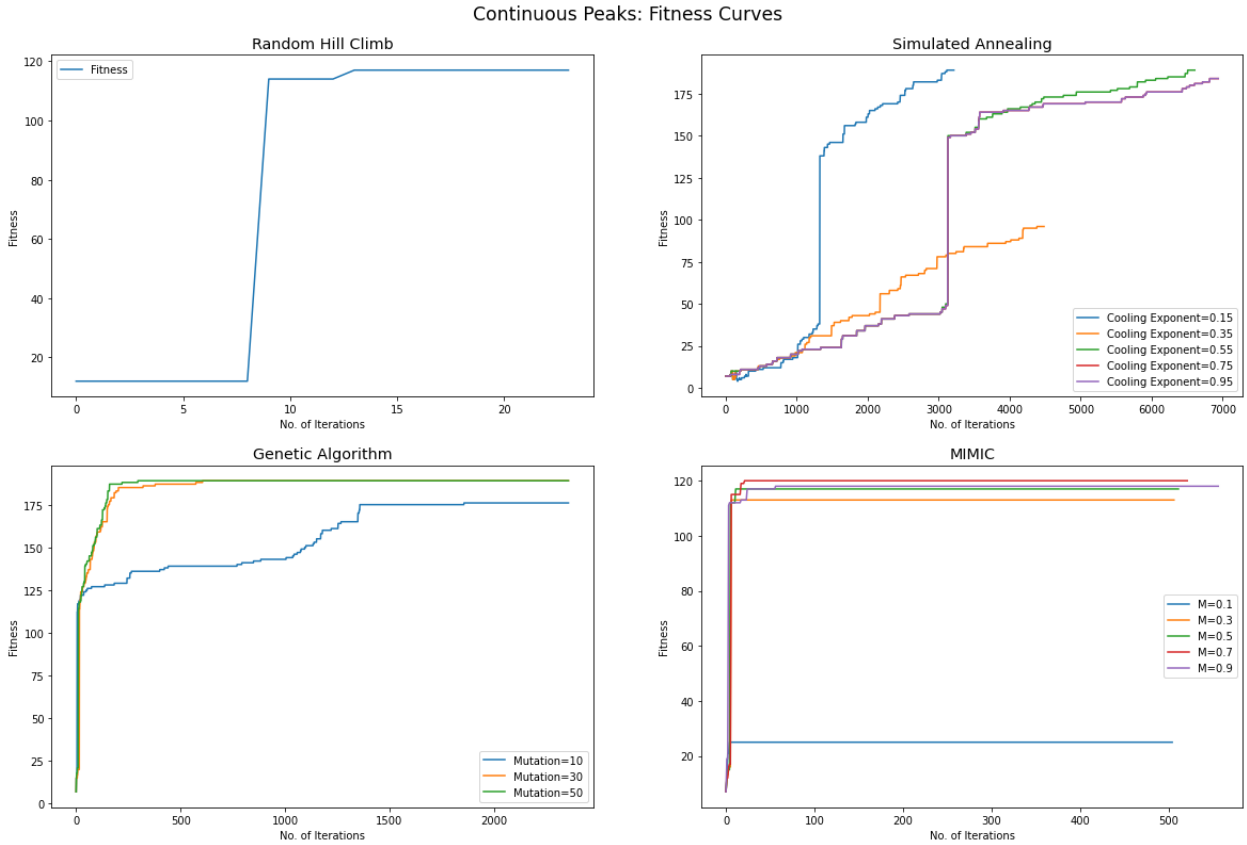
Both RHC and SA had a similar trajectory of performance over their iterations. RHC was able to gradually improve by randomly guessing routes, and SA was able to one-up RHC by applying some acceptance function over the solution space to make more "educated" guesses. Interestingly, the two best iterations of SA were of the second highest and the lowest cooling exponents (0.75 and 0.15). Since these cooling exponents essentially determine the rate of which possible routes are narrowed down, we see that in the first 1000 iterations, the higher cooling exponents achieved better fitness faster.

Finally, the poor performance of MIMIC is due to MIMIC requiring the structure of input dependencies. At a certain local optima, a single part of a route changing can lead to drastic differences in distance and fitness. These inputs of city orders also can't be designed as dependency trees.

Continuous Peaks

This problem was implemented with a bit length of 100, and there are multiple local optima. Below are the fit times of the optimization instances; averages across the same algorithm with different hyperparameters, and on the next page are the fitness results over number of iterations.

Randomized Optimization	Best Fitness	# Iter to Optima	Average Time (seconds)
Randomized Hill Climbing	117	13	4.736
Simulated Annealing	189 (CE = 0.15)	2165	0.395
Genetic Algorithm	189 (Mutation = 0.5)	296	10.340
MIMIC	120 (M = 0.7)	21	1971.243



An interesting contrast in the performance of these optimization techniques is that of MIMIC's M values. In the preceding TSP and in the Flip Flop problem we'll see later, the worst-performing M value is 0.9, the highest value tested. In this Continuous Peaks problem, the worst-performing M value by far was the lowest value tested of 0.1. That specific instance performed roughly 20% as well as all the other M values. This is due to the nature of the Continuous Peaks problem being structurally different compared to the other two. This makes sense as the joint probabilities of inputs is much more directly relevant to this Continuous Peaks problem. At M=0.1, the low dependency between parent and child nodes causes the fitness to plateau out at a low value as strong relationships can't be as easily determined.

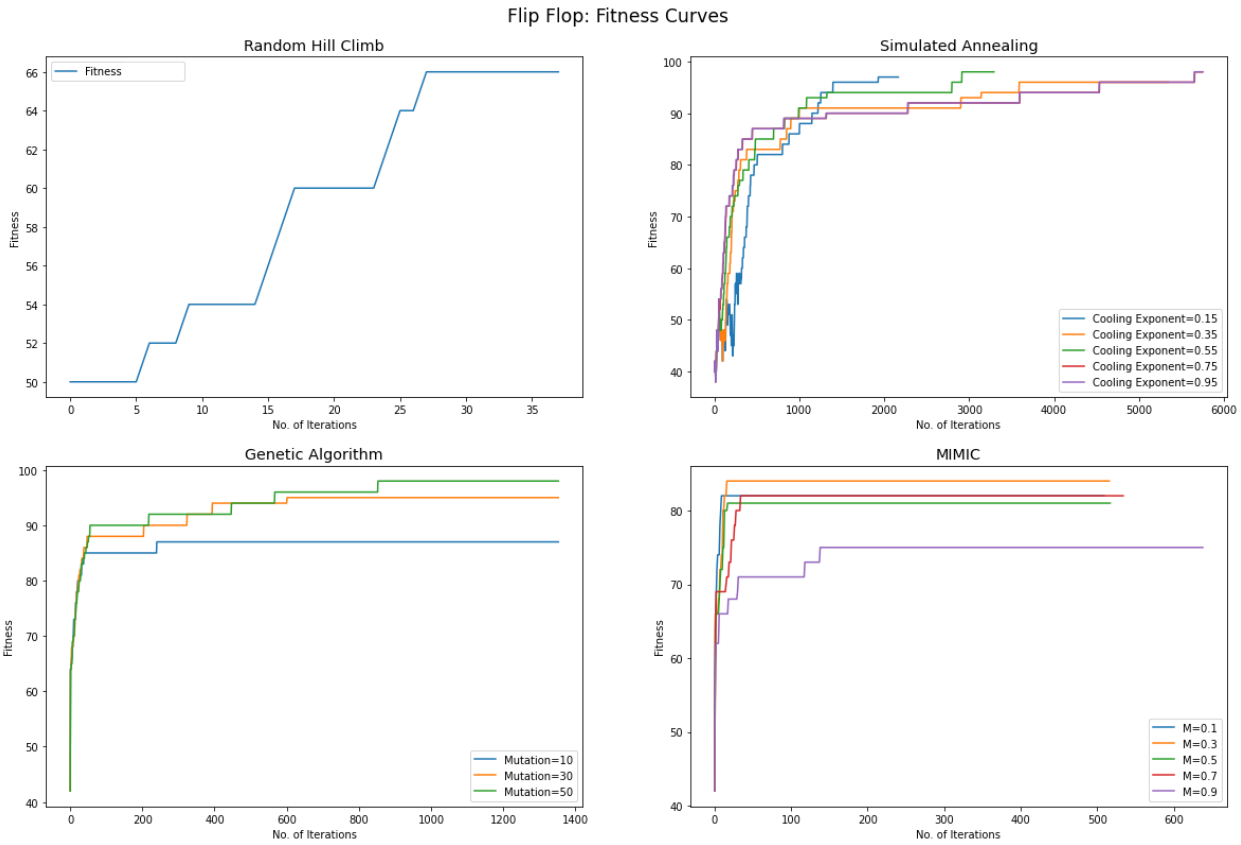
In terms of relative performance of MIMIC versus the other optimization algorithms, MIMIC with the exception of when paired with a low M value, reached their optima in much fewer iterations compared to the others at within the first 50 iterations. The best performing instance with an M value of 0.7 reached its optimum in only 21 iterations. Knowing this, MIMIC might not have to be trained for so many iterations as the computation involved per each iteration is high, as evident by MIMIC's consistently longer fit times. Despite this quick convergence however, MIMIC's peak fitness was only third best, being outperformed by both SA and GA.

Both SA and GA were able to get to the highest fitness values of 189, but GA was able to get there with much fewer iterations. With mutation rate set at 50%, optimal fitness was achieved by 300 iterations while the lowest number of iterations needed for SA with a cooling exponent of 0.15 was reached after ten times as many iterations.

Flip Flop

This problem counts the number of alternating bits in a bit string, so there are two possible global optima with fully alternating bits, i.e. [0, 1, 0, 1, ...]. This problem was implemented with a bit length of 100, so we know the best possible fitness score is 100.

Randomized Optimization	Best Fitness	# Iter to Optima	Average Time (seconds)
Randomized Hill Climbing	66	26	0.117
Simulated Annealing	98 (CE = 0.55)	2475	1.058
Genetic Algorithm	98 (Mutation = 0.5)	853	5.864
MIMIC	84 (M = 0.3)	16	1753.630



These results show that only GA was able to fully optimize this problem, achieving the maximum fitness score of 100 specifically with a mutation hyperparameter value of 50. This is not surprising, as these bits can only take two possible values, and since the goal is to get to alternating bits, the high mutation parameter allows the algorithm to quickly test opposite values of bits and achieve the optimal value within under 900 iterations. It is worth noting that no other optimization instance managed to achieve this optimal fitness score of 100.

RHC was implemented with a hyperparameter of 10 restarts, so although it only achieved a fitness score of 66, it is possible that allowing more restarts and thus total number of iterations, will allow it to eventually converge. This hypothesis is somewhat shown via the performance of SA, which has a similar peak-finding mechanism as RHC.

The effect of the cooling exponent is interesting in the fitness performance of each optimization instance. Within the first 1000-1500 iterations, all instances were able to achieve roughly 80-90 fitness, and the higher cooling exponents got there faster as more poorly performing bit strings are put below the acceptance threshold faster. None of these iterations managed to reach 100 fitness within 5000 iterations, but after reaching the 80-90 fitness range, the higher cooling exponent instance took longer to converge to its maximum fitness, potentially as the problem space is dimensionally complex, it hit some local optima and might have ruled out the global optima bit strings through the temperature decrease.

Finally, MIMIC's average performance across it's five instances with five M values outperformed RHC only by a little, with fitness scores around the 80s. Similar to TSP, the Flip Flop problem lacks the relationships between the 100 input variables, so modeling joint probabilities doesn't help with optimizing this problem. Within MIMIC though, the instance of M=0.9 was the worst performing optimization instance by far, topping out at 75. This shows that the dependency of child features to parent features in MIMIC's dependency trees should not be strong.

Discrete Optimization Problems: Summary

Below is the summary of the optimization algorithms' performance across the three discrete optimization problems:

Optimization Problem	**Highest Fitness**	**Fewest Iterations to Optima**
Traveling Salesman	GA @ M=0.3, (F=2588.496)	MIMIC @ M=0.9, 685 Iterations, F=10949.147
Continuous Peaks	GA @ M=0.5 (F=189)	MIMIC @ M=0.7, 21 Iterations, F=120
Flip Flop	GA @ M=0.5 (F=98)	MIMIC @ M=0.3, 16 Iterations, F=84

Consistent across the three problems, the algorithm that achieved the best fitness is GA with a 50% mutation rate. Worth noting is that SA also achieved the same fitness scores for Continuous Peaks and Flip Flop but after many more iterations. RHC performed the worst for all three problems, and MIMIC came in third best for all three as well. However, MIMIC was able to get to its best fitness score in the fewest number of iterations. It's complex mechanism of calculating joint probabilities helped it to quickly achieve convergence, but not to the global optima due to the unstructured nature of the input variables in the problems chosen.

Part 2: Neural Network Classification Problem

For this section, a Credit Card Fraud dataset on 1,492 European transactions was used to test the performance of randomized optimization algorithms against the performance of backpropagation. In this dataset, there were 1420 transactions labeled as non-fraudulent and 72 transactions labeled as fraud, making it highly imbalanced. For up to 5000 iterations, each of RHC, SA and GA were run, as well as 5000 iterations/epochs of backpropagation. To allow for more direct comparison, we first look for the best hyperparameters for SA and GA. The following hyperparameters were tested:

- Simulated Annealing:
    - Cooling exponents [0.15, 0.35, 0.55, 0.75, 0.95]
- Genetic Algorithm: Population = 100, Crossover = 30
    - Mutation [10, 30, 50]

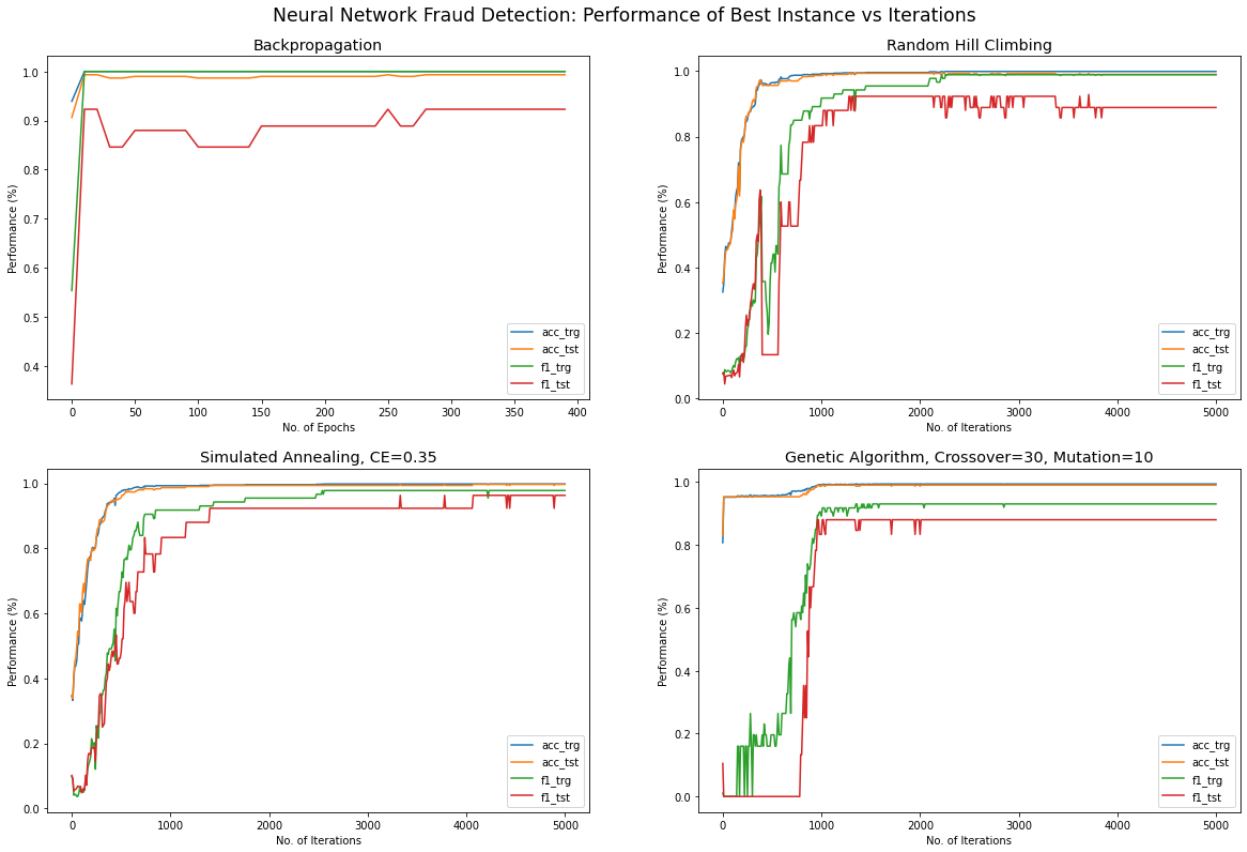
Randomized optimization hyperparameter tuning results:

SA Cooling Exponent	Final F1 Score (%)	Best F1 Score (%)	# Iterations to Highest F1 Score
0.15	0.880	0.923	830
0.35	0.963	0.963	3330
0.55	0.923	0.923	3250
0.75	0.889	0.923	1570
0.95	0.889	0.923	1930

GA Mutation	Final F1 Score (%)	Best F1 Score (%)	# Iterations to Highest F1 Score
10	0.880	0.880	960
30	0.889	0.923	1910
50	0.857	0.929	1180

For SA, lower cooling exponents outperform higher cooling exponents, probably due to the high number of parameters to adjust values for and the complexity of a neural network not allowing for many solutions to be ruled out with low acceptance thresholds. For GA, the results were mixed across the three mutation values tested, but I decided to go ahead with a mutation of 10 due to its relatively quick convergence and its final F1 score was only marginally lower than the scores of the other mutation values. Below is a summary of the performance of the four algorithms.

Optimization Algo	Final F1 Score (%)	# Iterations to Highest F1 Score	Time to Highest F1 Score (s)	Total Time for 5000 iterations (s)
Backpropagation	0.923	280	1.315	1.768
Random Hill Climbing	0.889	3710	8.539	11.340
Simulated Annealing (0.35)	0.963	3330	7.643	11.557
Genetic Algorithm (10)	0.880	960	144.436	295.021



From the graphs above examining how each algorithm converges to their optima as well as the preceding summary table of results, we have several observations:

Speed of Convergence

Backpropagation far outperformed the other algorithms in terms of how quickly it was able to converge to an optima, doing so in 280 iterations. The next best algorithm in terms of convergence was GA, converging in 960 iterations. Both RHC and SA, the algorithms that have peak-finding mechanisms, took much longer to converge, needing at over 3000 iterations to converge. This is due to the high complexity of the solution space of a neural network.

Achieved Optima

SA was able to achieve the overall highest F1 score at 96.3%, followed by Backpropagation at 92.3%, RHC at 88.9% then GA at 88.0%. Because SA outperformed backpropagation, backpropagation likely fell into a local optima due to the nature of gradient descent. SA having an element of randomness allowed it to escape this local optima trap that backpropagation seemingly fell into.

Performance Stability

Overall, test metrics were much less stable compared to train metrics, as expected as these algorithms first seek to learn the training set and only then gradually generalize to the testing set. Backpropagation looked to be the most stable, but it could be due to the fact that it was only trained for about 400 epochs compared to the 5000 iterations the other algorithms optimized over. RHC, as expected, had the most instability in the first 1000 iterations due to its random restarting of peak finding, especially in its test metrics. GA was extremely unstable over the first 900 iterations, but remained very stable once it reached convergence, as

by then, only the solutions that were performing well were the ones that were left to be crossed over and mutated.

Bias-Variance Tradeoff

All optimization algorithms had overfitting issues, though all to similar degrees. This can likely be combated through tuning the hyperparameters of the underlying neural network being used. We see that the training accuracy scores consistently approach a perfect 100%. This neural network optimizes for binary cross-entropy, not F1 score or recall. And on this imbalanced dataset where only 4.8% of the total instances are fraud transactions, the model isn't effectively optimizing for detecting the minority class.

References

- Hayes, G. (2019). mlrose: Machine Learning, Randomized Optimization and SEarch package for Python. https://github.com/gkhayes/mlrose. Accessed: 10 Oct 2021
- Pushkar (2018). Pushkar/ABAGAIL. https://github.co/pushkar/ABAGAIL. Accessed: 10 Oct 2021
- Credit Card Fraud Detection Dataset. https://www.kaggle.com/isaikumar/creditcardfraud Accessed 10 Sep 2021