

Exploring Random Optimization (RO): Evaluating RO Algorithms for Wine Quality Prediction

Marcus Tan
mtan75@gatech.edu

Abstract — This paper explores four random optimization algorithms—Random Hill Climbing, Simulated Annealing, Genetic Algorithm, and MIMIC. Each algorithm is examined in detail, such as unique features and formulated hypotheses. Three of these algorithms are then applied to the apple quality dataset, offering insights into their efficacy in the context of neural network.

1 INTRODUCTION

In recent years, there has been a notable surge in interest surrounding the development of random optimization algorithms, potentially fueled by the advancements in computing power (Smith, Johnson, & Williams, 2020). Moreover, there is a rising interest in applying certain random optimization algorithms, notably the Genetic Algorithm, within the domain of machine learning. This paper aims to contribute to the evolving field by researching four random optimization algorithms: Random Hill Climbing, Simulated Annealing, Genetic Algorithm, and MIMIC. Subsequently, we will compare three of these algorithms (Random Hill Climbing, Simulated Annealing, Genetic Algorithm) against a neural network model created through backpropagation.

This paper is divided into four sections: (1) an introduction of the algorithms and dataset, along with their characteristics and hypotheses; (2) an exploration of algorithm performance in three random optimization problems; (3) an analysis of dataset performance using both backpropagation and random optimization algorithms; and (4) concluding with a comparison of the algorithms and additional analysis that could be conducted with more time.

2 RANDOM OPTIMIZATION ALGORITHM AND DATASET USED

2.1 Algorithm Introduction

[Random Hill Climbing (RHC)] RHC is a search algorithm that begins with a random solution and iteratively improves by making incremental changes and occasional restarts. It explores random initial guesses, moving towards higher fitness, and is very useful in navigating convex problem spaces due to its simplicity.

[Simulated Annealing (SA)] SA is a probabilistic optimization algorithm that begins with an initial solution, exploring neighboring solutions with decreasing probability for accepting worse outcomes. This approach helps the algorithm navigate solution spaces effectively, aiming to find the global optimum by balancing exploration and exploitation.

[Genetic Algorithm (GA)] Genetic Algorithm (GA) is an optimization method inspired by natural selection. It operates with a population of potential solutions, using genetic operations like selection, crossover, and mutation across successive generations. This approach mimics natural evolution, enabling efficient exploration of solution spaces to find optimal or near-optimal solutions for complex problems.

[MIMIC] MIMIC, or Mutual Information Maximization for Input Clustering, is a probabilistic optimization algorithm. It employs a statistical model to capture dependencies among variables in the problem space. MIMIC iteratively refines its model through the selection of promising solutions, ultimately converging to an optimal or near-optimal solution. This algorithm is particularly effective in handling complex problems with intricate dependencies among variables.

2.2 Dataset Introduction

[Dataset Apple Quality] The Apple Quality dataset "[apple quality.csv](#)" (Elgiriye withana, 2023) stands out as one of the cleanest datasets available for our analysis. It presents several favorable qualities, including being a binary class problem, low correlation across columns, and a balanced distribution of classes. However, its selection as our dataset is driven by a specific challenge: the presence of outliers.

[Outliers] The apple quality has several outliers present in the dataset. Specifically, columns such as weight and crunchiness exhibit a significant outlier condition, impacting 1-3% of the dataset. Outliers are identified as data points falling outside the 1.5 Interquartile Range. This dataset was intentionally chosen to focus on the outlier problem, providing a rigorous test for the ability of random optimization algorithms to navigate towards a global optimum.

2.3 Hypothesis – How do we define best and which algorithm is hypothesized to perform the best?

The definition of "best" in this context hinges on achieving the highest fitness value in the shortest iteration. While metrics like efficiency, accuracy, and wall clock times are relevant, the primary criterion is maximizing fitness in the shortest iteration possible. We will additionally evaluate function evaluations, wall clock time, and fitness per size to comprehensively assess algorithm performance.

Given the proven efficacy over the years, Genetic Algorithm stands out as a widely favored top-performing random optimization algorithm, especially in the realm of machine learning. As such, I hypothesize that this is the top algorithm in both random optimization and machine learning contexts.

3 DEEP DIVE INTO RANDOM OPTIMIZATION ALGORITHMS

3.1 Introduction to Optimization Problems – Why were these problems selected?

Before delving into their application in machine learning, we will assess the effectiveness of the algorithms by subjecting them to three random optimization problems: the N Queens problem, the Traveling Salesman Problem (TSP), and the Continuous Peak Problem (CPP). To ensure fair comparison across all algorithms, each of these algorithms will rely on the average of 5 seeds to generate the fitness values.

[N Queens Problem: GA Performs Best] The N Queens problem is a classic chessboard puzzle where the objective is to place N chess queens on a N×N chessboard in such a way that no two queens threaten each other. In other words, no two queens should share the same row, column, or diagonal. The challenge lies in finding a configuration that satisfies these constraints, and the problem is a well-known example in combinatorial optimization and computer science. The N Queens problem was selected because there could be multiple global solutions and hence, this problem will be useful in helping us identify which algorithm would achieve a state with the optimal fitness in the shortest iteration possible.

[Travelling Salesman Problem (TSP): MIMIC performs best at size = 8] TSP is a classic optimization challenge where the goal is to determine the most efficient route that visits a set of cities exactly once and returns to the starting city. The objective is to minimize the total distance or cost of the tour, or maximize the reciprocal of the total distance. The challenge lies in finding the optimal permutation of city visits out of all possible combinations, considering the distances between each pair of cities. The Traveling Salesman Problem is selected as a problem because of its NP-hard nature and hence, it's impossible to test all combinations. The nature of this problem set will allow us to see which algorithm achieves highest fitness (in this case minimal distance), and in the shortest iteration possible.

[Continuous Peak Problem (CPP): SA Performs Best] CPP is an optimization challenge where the goal is to locate the peak of a continuous and multi-dimensional landscape. In this context, "peak" refers to the highest point in the function's domain. The challenge is to identify the input values that lead to the maximum value of the objective function, and might be helpful in evaluating models that exploit vs explore. CPP is chosen because it contains many local optima, and it helps to test algorithms capability in exploiting vs exploring.

3.2 Evaluating Random Optimization Algorithms through N Queens

Optimal hyperparameter selection is pivotal for meaningful comparisons across the four algorithms. Initial analysis focused on a base case for the N Queens problem, with N set to 6. In instances with multiple optimal solutions, priority was given to those with lower compute resource requirements. Additionally, despite being a minimization problem, the approach was converted into a maximization problem by emphasizing the maximization of non-attacking queens.

[Optimal RHC Hyperparameters; Wall Clock Time: 0.3s] In the optimization of Random Hill Climbing, we fine-tuned the hyperparameter associated with the number of restarts. We experimented with a range from 0 to 10 restarts between 200 and 600 iterations, we observed that the optimal restart value is 3.

# of Restarts	0	1	2	3	4
Fitness	14	13	13	15 (optimal)	14

[Optimal SA Hyperparameters; Wall Clock Time: 0.3s] In optimizing Simulated Annealing, we fine-tuned the hyperparameter linked to the temperature. The choice of the optimal temperature was guided by the achieved fitness levels. In our experimentation, we explored a temperature range from 10000 to 1×10^{12} , and 100 to 500 iterations. Our observations indicated that the optimal temperature for this scenario is 10,000.

Temperature	10,000	1,000,000	100,000,000 (1×10^8)	10,000,000,000 (1×10^{10})	1,000,000,000,000 (1×10^{12})
Fitness	13 (optimal)	10	13	10	10

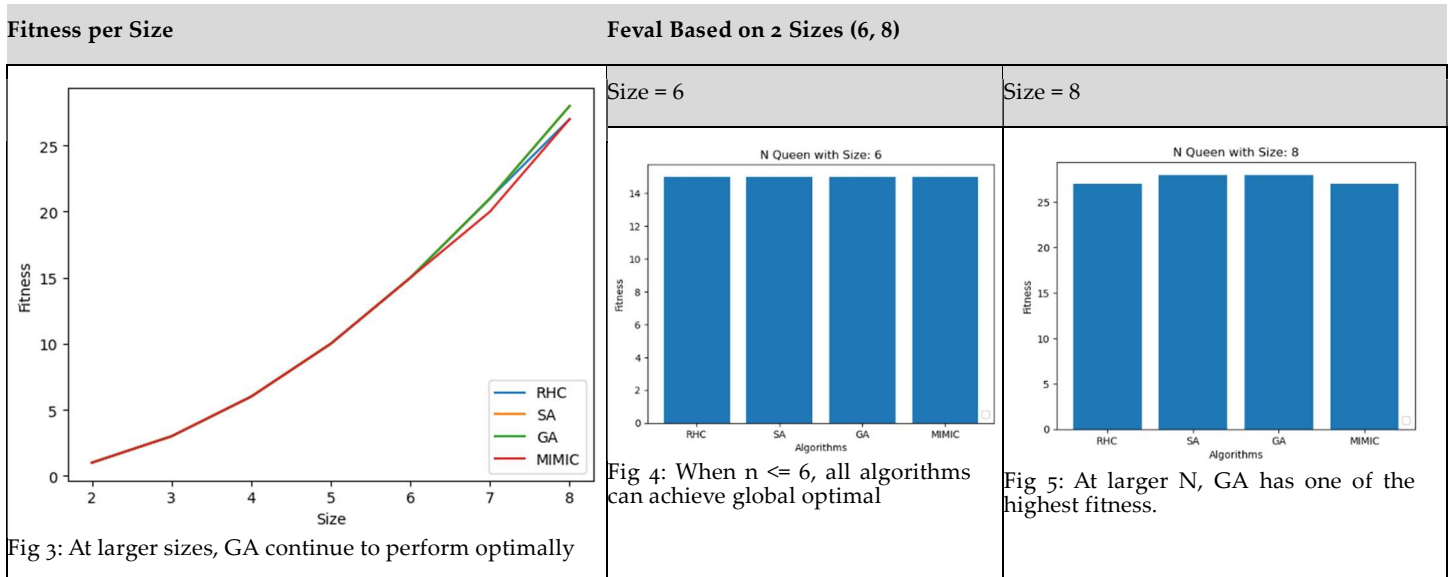
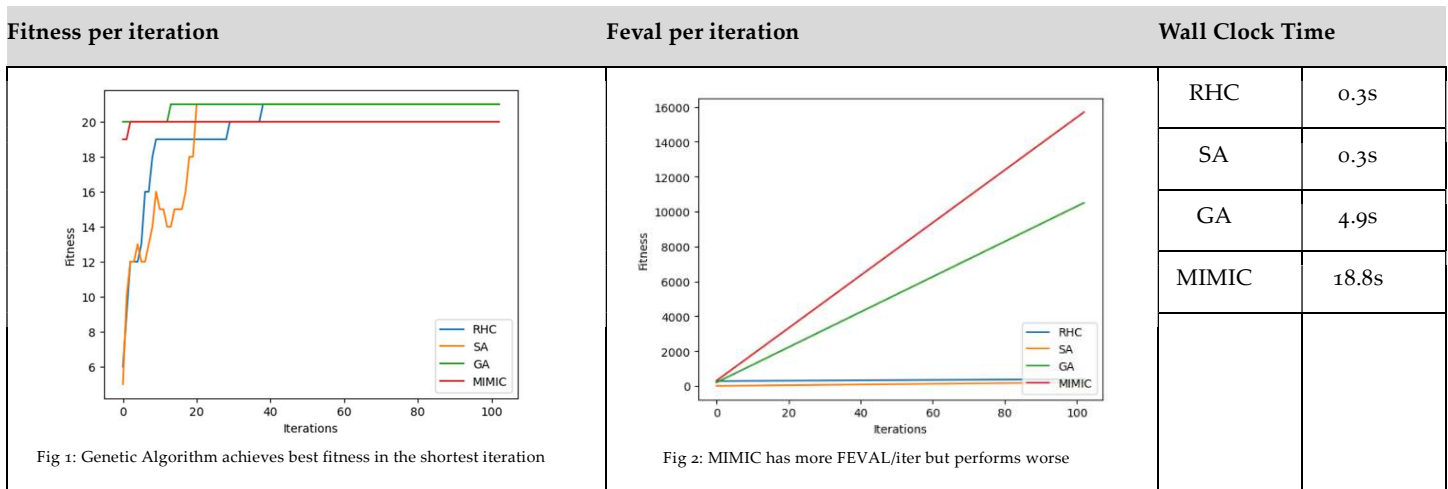
[Optimal GA Hyperparameters; Wall Clock Time: 4.9s] In the optimization of the Genetic Algorithm for the N Queens problem with a size of 6, we conducted fine-tuning for both population size and mutation rates. Through this process, we identified that the optimal configuration is a population size of 100 and a mutation rate of 0.2, achieving the highest fitness levels.

Population Size	50	100	150	200	250	300	350	400
Mutation Rates (0.1)	14	14	14	15	15	15	15	14
Mutation Rates (0.2)	14	15 (optimal)	14	15	15	14	15	15
Mutation Rates (0.3)	14	14	14	15	15	15	15	15

[Optimal MIMIC Hyperparameters; Wall Clock Time: 18.8s] Similar to the Genetic Algorithm, we fine-tuned two hyperparameters—population size and the keep_pct. Our observations revealed that the optimal configuration for these hyperparameters is a population size of 150 and a retention percentage of 0.4, achieving the highest fitness levels.

Population Size	50	100	150	200	250
Keep_pct (0.2)	13	14	14	14	15
Keep_pct (0.3)	13	14	14	14	15
Keep_pct (0.4)	14	14	15 (optimal)	14	15
Keep_pct (0.5)	14	14	15	14	14

[Overall Analysis on N Queen] Using the hyperparameters determined earlier, we visualized and compared the results of the four algorithms.



In summary, the charts reveal key insights pertaining to algorithmic performance. For N-queens with a size of 6, Genetic Algorithm (GA) stands out as the algorithm achieving the highest fitness in the shortest iterations, while MIMIC performs relatively slower (Fig 1). Although GA is not the fastest, it consistently attains the highest fitness levels across varying sizes for N-queens (Fig 3, Fig 5). Notably, GA accomplishes this without requiring an excessive number of function evaluations, as compared to MIMIC (Fig 2). MIMIC, on the other

hand, demands the most function evaluations, potentially due to 2 reasons. They are: (1) the higher population sizes and percentage of population retention and (2) MIMIC maintains a probabilistic model of the solution space, which involves estimating a joint probability distribution over the problem variables; this model needs to be updated iteratively, leading to increased computational demands. Both of these reasons ultimately impacts the performance in wall clock time for MIMIC.

As discussed in the problem introduction, the N Queens problem was selected because there could be multiple global solutions and hence, this problem will be useful in helping us identify which algorithm would achieve the optimal fitness in the shortest iteration possible. In this problem, Genetic Algorithms (GA) tend to perform well consistently across different sizes of N due to their inherent adaptability and exploration-exploitation balance. GA leverages a population-based approach, maintaining a diverse set of candidate solutions through generations. This diversity allows GA to explore a broad solution space efficiently. Additionally, the crossover and mutation operations in GA facilitate the exchange and modification of genetic material, enabling it to adapt to different problem sizes and complexities.

3.3 Evaluating Random Optimization Algorithms through TSP

Hyperparameters were tuned for the four algorithms in the Traveling Salesman Problem (TSP) with 8 nodes to enable a comparative analysis. Notably, the goal in TSP is typically to minimize the distance traveled, but for our study, we transformed it into a maximization problem by taking the reciprocal of the fitness value.

[Optimal RHC Hyperparameters; Wall Clock Time: 0.1s] We tested a range of restart values from 1 to 5 to identify the value most likely to yield maximum fitness.

# of Restarts	1	2	3	4	5
Fitness	104 (optimal)	104	104	104	104

[Optimal SA Hyperparameters; Wall Clock Time: 0.3s] We tested a range of temperature list from 1 to 10 billion to identify the values that are likely to maximize fitness.

# of Temperature	1	10000	1000000	100000000	10,000,000,000
Fitness	104 (optimal)	130	199	199	199

[Optimal GA Hyperparameters; Wall Clock Time: 3.8s] Similar to the N Queens problem, we will fine-tune the population size and mutation rate for the Genetic Algorithm. We explored a range of population sizes from 2 to 300 and mutation rates from 0.2 to 0.5.

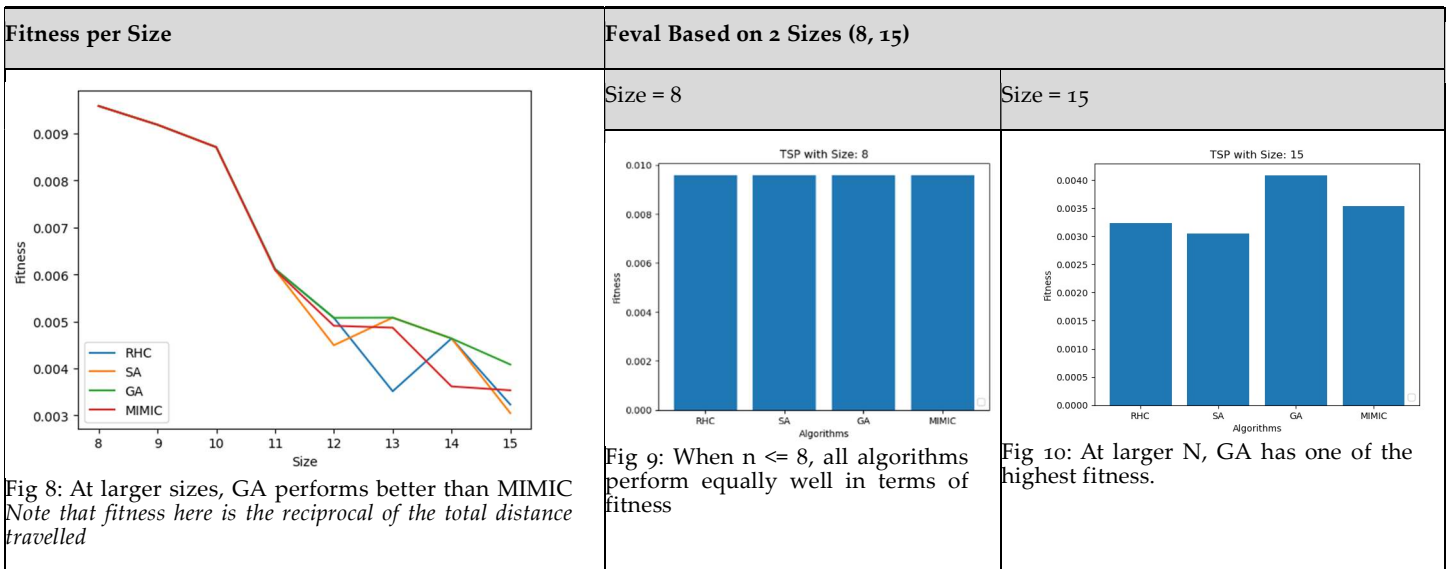
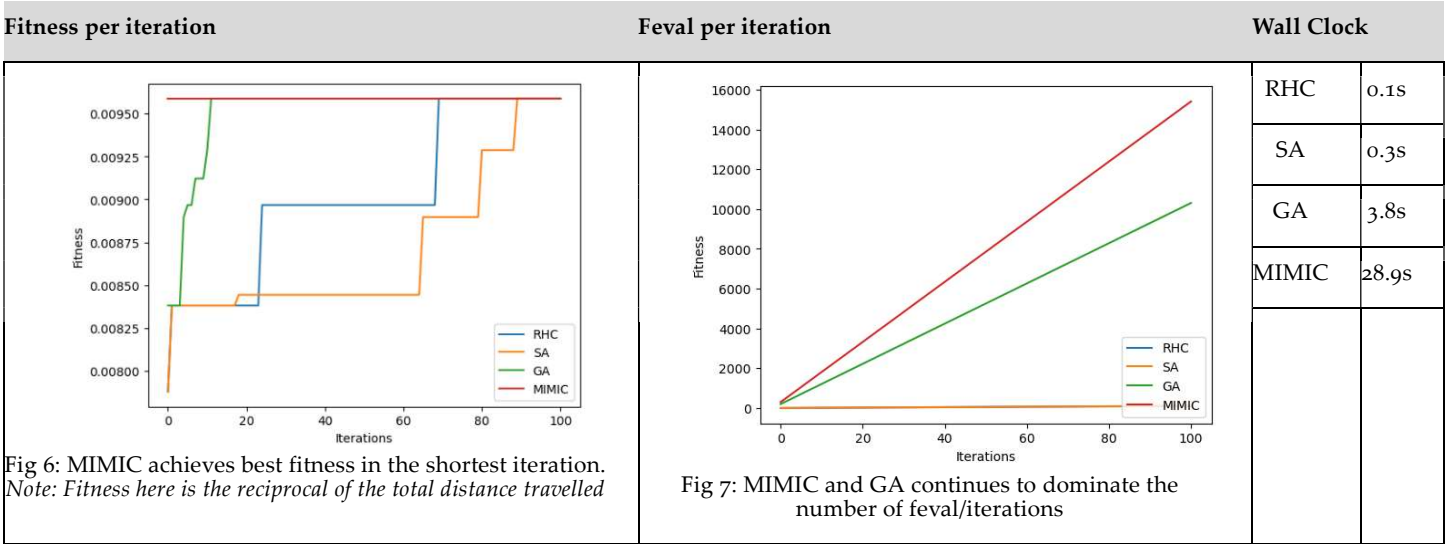
Population Size	2	100	200	300	400
Mutation Rates (0.2)	133	104 (optimal)	104	104	104
Mutation Rates (0.3)	125	104	104	104	104
Mutation Rates (0.4)	113	104	104	104	104

[Optimal MIMIC Hyperparameters; Wall Clock Time: 28.9s] In tuning the MIMIC algorithm, we systematically varied the population size from 50 to 250 individuals and explored the retention rate (keep_pct) within the range of 0.1 and 0.5. This thorough examination sought to pinpoint the optimal hyperparameter settings for improved performance in addressing the specific optimization problem.

Population Size	50	100	150	200	250
Keep_pct (0.1)	117	119	117	107	107

Population Size	50	100	150	200	250
Keep_pct (0.2)	135	119	104 (optimal)	104	104
Keep_pct (0.3)	140	118	107	104	104

[Overall Analysis on TSP] Using the hyperparameters determined earlier, we visualized and compared the results of the four algorithms.



Several key insights emerge from the conducted experiments: MIMIC exhibits strong fitness performance for smaller node sizes, and generally, Genetic Algorithm (GA) and MIMIC perform comparably for TSP when the size is below 7. However, a notable shift occurs at size = 8, where MIMIC quickly discovers the optimal solution and maintains a plateau thereafter. Similar to N Queens, MIMIC also requires the largest FEvals/iteration and takes longer to run (Fig 7).

Considering our criteria for the best algorithm—maximizing fitness in the shortest iterations—it becomes evident that MIMIC excels specifically for size = 8. Despite this achievement, MIMIC's convergence is notably slower, at times almost 30 times slower than Random Hill Climbing (RHC) and Simulated Annealing (SA), mirroring observations made in the N Queens problem. Interestingly, as the size of N increases, GA demonstrates superior fitness compared to MIMIC.

RHC's Volatility

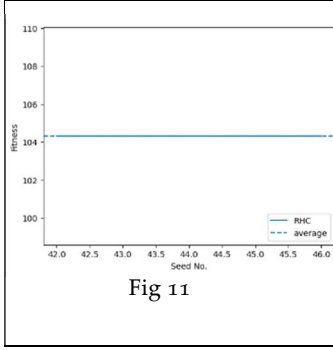


Fig 11

SA's Volatility

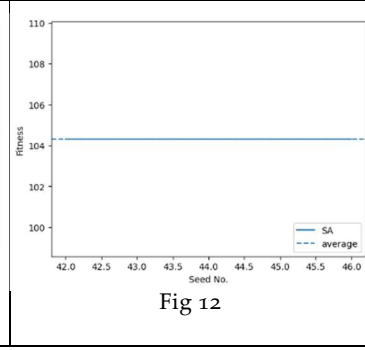


Fig 12

GA's Volatility

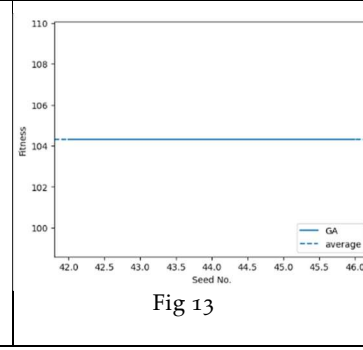


Fig 13

MIMIC's Volatility

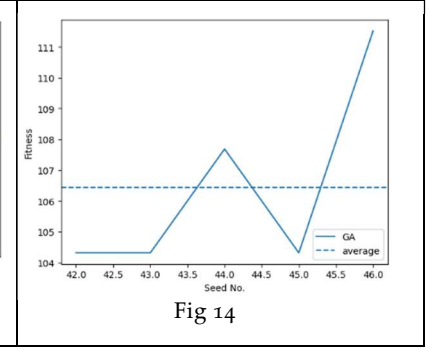


Fig 14

This prompts a question regarding MIMIC's exceptional performance at size = 8 and raises a hypothesis that the observed success might be attributed to the randomness inherent in the chosen optimization seeds. Indeed, upon further analysis (Fig 11- Fig 14), it is evident that MIMIC experiences high volatility in results at larger sizes, potentially contributing to its sporadic outperformance at specific dimensions.

3.4 Evaluating Random Optimization Algorithms through Continuous Peak Problem

In alignment with our approach for other problems, the initial step involved tuning all algorithms. The preliminary analysis concentrated on a base case for Continuous Peak Problem (CPP), with N set to 100. In cases where multiple optimal solutions were identified, priority's given to those with lower compute requirements.

[Optimal RHC Hyperparameters; Wall Clock Time: 0.7s] We tested a range of restart values from 1 to 5 to identify the value most likely to yield maximum fitness.

# of Restarts	1	2	3	4	5
Fitness	15	35 (optimal)	18	28	12

[Optimal SA Hyperparameters; Wall Clock Time: 14.9s] We tested a range of temperature list from 10 to 10 billion to identify the values that are likely to maximize fitness, while setting decay schedules to 0.99.

Temperature	10	100	1,000	10,000	100,000
Fitness	185	188	100	189 (optimal)	100

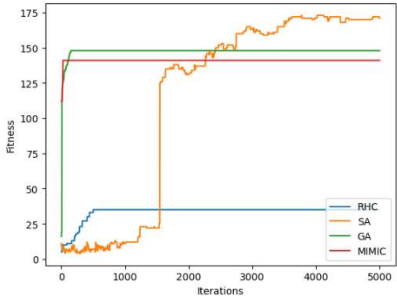
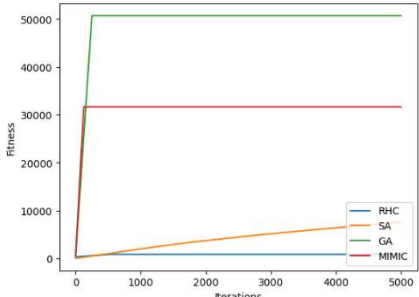
[Optimal GA Hyperparameters; Wall Clock Time: 27s] Similar to the N Queens problem, we will fine-tune the population size and mutation rate for the Genetic Algorithm. We explored a range of population sizes from 50 to 250 and mutation rates from 0.1 to 0.3.

Population Size	50	100	150	200	250
Mutation Rates (0.1)	132	144	156	148	184
Mutation Rates (0.2)	133	177	157	128	148
Mutation Rates (0.3)	135	185	165	187	161

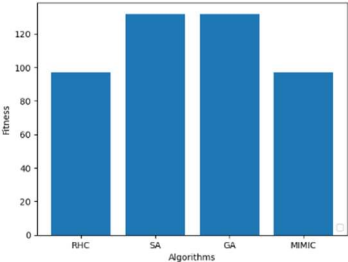
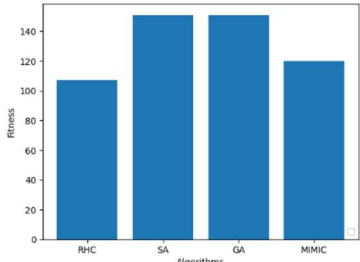
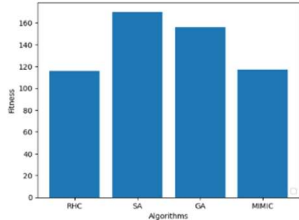
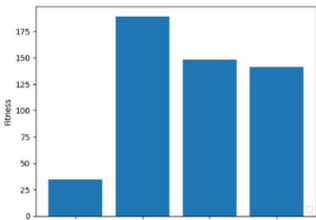
[Optimal MIMIC Hyperparameters; Wall Clock Time: 3954s] While fine-tuning the MIMIC algorithm, we methodically adjusted the population size across a range from 50 to 250 individuals. Additionally, we explored the retention rate (keep_pct) within the range of 0.1 to 0.5.

Population Size	50	100	150	200	250
Keep_pct (0.1)	113	125	121	126	128
Keep_pct (0.3)	20	120	129	131	136
Keep_pct (0.5)	130	122	130	130	141 (optimal)

[Overall Analysis on CCP] Using the hyperparameters determined earlier, we visualized and compared the results of the four algorithms.

Fitness per iteration	Feval per iteration	Wall Clock	
 <p>Fig 15: SA achieves best fitness at the end of 5000 iterations</p>	 <p>Fig 16: MIMIC and GA's Feval/iteration continues to dominate other algorithms</p>	RHC	0.7s
		SA	14.9s
		GA	27s
		MIMIC	3954s

Fitness/Size Based on 4 Sizes (80, 90, 100, 110)

<p>Size = 80</p>  <p>Fig 17: When size = 80, SA and GA performs equally</p>	<p>Size = 90</p>  <p>Fig 18: When size = 90, SA continues to perform equally to GA.</p>
<p>Size = 100</p>  <p>Fig 19: SA outperforms other algorithms</p>	<p>Size = 110</p>  <p>Fig 20: SA continues to outperform other algorithm at larger sizes</p>

As with N Queens and TSP problem, the MIMIC algorithm exhibits a notably slower iteration pace compared to the other three algorithms. In this specific scenario, the SA algorithm achieved the best fitness score within 5000 iterations, while other algorithms reached a plateau before 1000 iterations (Fig 15). Among all the algorithms, MIMIC has the most number of Feval/iteration and (Fig 16) performed the least favorably in terms of (1) fitness score and (2) running time for this problem. Throughout the experimental process, it has become evident that parameter selection for SA is highly crucial, as it could result in vastly different outcomes. For instance, changing the initial temperature for SA to 1 yield significantly lower result than algorithms like RHC.

This experiment has also revealed SA's impressive capability to escape local optima, especially with higher initial temperatures. Higher initial temperature allows SA to explore before exploiting, enabling it to break free from local optima and achieve superior results, at a range of sizes (Fig 17 - 20).

4 APPLYING RANDOM OPTIMIZATION TO MACHINE LEARNING

[Back Propagation Tuning; Time to Train/Test: 17s] Similar to the previous random optimization problems, we initially fine-tuned hyperparameters for our neural network by experimenting with various activation functions, layers, nodes, and learning rates. Our chosen metric throughout this section for assessing fitness is the recall score for the validation set (cross validation: 2 fold), ensuring consistency with the metrics used in earlier assignment. Sample of the hyperparameters that we experimented is listed in the tables below.

Nodes: 64,64,64 Activation: Sigmoid LR: 0.0001	Nodes: 64,64,64,64 Activation: ReLU LR: 0.00001	Nodes: 128,128,128 Activation: Tanh LR: 0.001	Nodes: 64,64,64 Activation: Tanh LR: 0.0001	Nodes: 64,64,64 Activation: Sigmoid LR: 0.00001	Nodes: 64,64,64,64 Activation: Tanh LR: 0.0001	Nodes: 128,128,128,128 Activation: ReLU LR: 0.0001
Val Recall: 0.82	Val Recall: 0.80	Val Recall: 0.82	Val Recall: 0.87 (optimal)	Val Recall: 0.76	Val Recall: 0.82	Val Recall: 0.50
Train Recall: 0.84	Train Recall: 0.98	Train Recall: 0.99	Train Recall: 0.99	Train Recall: 0.76	Train Recall: 0.98	Train Recall: 0.50

[RHC Tuning; Time to Train & Test: 22s] After establishing the structure of the neural network, we proceeded to optimize parameters using Randomized Hill Climbing (RHC) by adjusting the number of restarts.

Restarts = 0	Restarts = 1	Restarts = 2	Restarts = 3	Restarts = 4	Restarts = 5
Val Recall: 0.47	Val Recall: 0.55	Val Recall: 0.55	Val Recall: 0.57 (optimal)	Val Recall: 0.50	Val Recall: 0.49
Train Recall: 0.47	Train Recall: 0.55	Train Recall: 0.55	Train Recall: 0.57	Train Recall: 0.51	Train Recall: 0.49

[SA Tuning; Time to Train & Test: 11s] For simulated annealing, I observed that altering only the init_temp and decay rates had no impact on validation scores. To ensure a balance between exploiting and exploring, various combinations of schedules and learning rates were tested, as outlined below.

Init_temp = 10000 Decay = 0.9 Learning Rate = 0.9	Init_temp = 10000 Decay = 0.8 Learning Rate = 0.8	Init_temp = 100000000 Decay = 0.7 Learning Rate = 0.7	Init_temp = 1000000000 Decay = 0.6 Learning Rate = 0.6	Init_temp = 1000000 Decay = 0.9 Learning Rate = 0.5	Init_temp = 100000000 Decay = 0.8 Learning Rate = 0.4
Val Recall: 0.52	Val Recall: 0.53 (optimal)	Val Recall: 0.50	Val Recall: 0.48	Val Recall: 0.46	Val Recall: 0.47
Train Recall: 0.53	Train Recall: 0.53	Train Recall: 0.50	Train Recall: 0.49	Train Recall: 0.46	Train Recall: 0.47

[GA Tuning; Time to Train & Test: 55210s] Similar to previous optimization problems, we experimented with varying the population size and mutation rate for the genetic algorithm, as depicted below.

Pop Size = 700 Mutate Prob = 0.7	Pop Size = 500 Mutate Prob = 0.5	Pop Size = 300 Mutate Prob = 0.3	Pop Size = 100 Mutate Prob = 0.1
Val Recall: 0.61	Val Recall: 0.64 (optimal)	Val Recall: 0.59	Val Recall: 0.62
Train Recall: 0.62	Train Recall: 0.64	Train Recall: 0.59	Train Recall: 0.62

[Overall Analysis on Apple Quality Dataset] After fixing the optimal backpropagation architecture and incorporating specific hyperparameters for each algorithm, we assessed the recall score for each algorithm. Subsequently, the fitness at each maximum iteration is being plotted to evaluate their performance in predicting apple quality. Although backpropagation doesn't reach its optimum in the shortest iterations (Fig 21), it consistently outperforms random optimization (Fig 23) and completes the runs in the shortest duration (Fig 22).

Backpropagation tends to perform better than random optimization algorithms when dealing with well-structured optimization problems. It excels in datasets that have visible gradients (Freedman, 2019).

$$\theta_{ij} = \theta_{ij} - \alpha \cdot \frac{\Delta \theta_j}{\Delta \theta_{ij}}$$

where θ_{ij} represents the weights between neuron i in one layer and neuron j in

the subsequent layer, J is the loss function and α is the learning rate.

Using the above equation, having a “visible gradient” means that the values of $\frac{\Delta \theta_j}{\Delta \theta_{ij}}$ are large or significant indicating that the loss is changing notably concerning the specific weight. This visibility suggests that the optimization process is making meaningful adjustments to the weights, and the model is learning effectively.

As evident from assignment 1, apple quality dataset exhibits this characteristic, where backpropagation is very effective; assignment 1 achieved Recall scores of >80% on Apple Quality as well. Indeed, additional fine-tuning of the backpropagation model from Assignment 1 has unveiled that recall scores can be further improved by excluding outliers.

Fig 21: Fitness (Recall) per Iteration

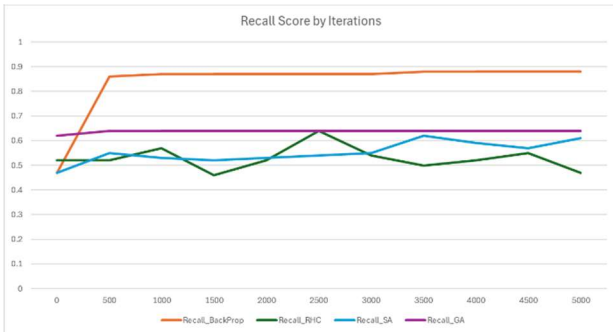


Fig 22: Wall Clock Time

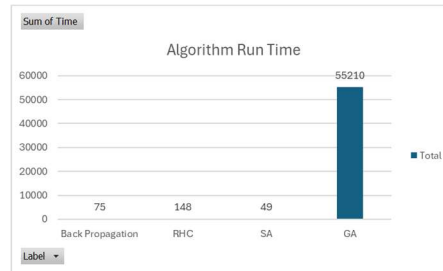
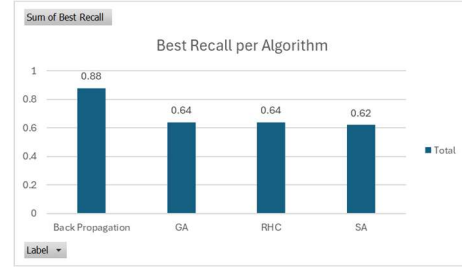


Fig 23: Best Recall per Algorithm



5 CONCLUSION & DISCUSSION OF ALGORITHMS

In summary, this report presents insights into the performance of various algorithms across three optimization problems and their application to a machine learning scenario. Each algorithm exhibits unique strengths and weaknesses, making them suitable for different contexts based on factors like computational resources and time constraints. The table below encapsulates key findings related to these algorithms.

	Pros	Cons
RHC	Easy to tune with only one parameter, fast execution.	Limited generalization, particularly for complex problems like CCP.
SA	Conceptually easy to explain, effective in approximating global optima	Parameter tuning for more challenging problems can be tough, as the optimal initial temperature could be in the exponential
GA	Generally fast with competitive results, except in machine learning problems.	Involves tuning multiple hyperparameters, challenging for new users.
MIMIC	Generally superior results compared to RHC and SA.	Extremely slow, requires tuning multiple hyperparameters.

While the analysis provides valuable insights into the algorithms, there are opportunities for further refinement. Given more time, it would be beneficial to explore the impact of removing outliers from the apple quality dataset on algorithm performance. Additionally, investigating other random optimization problems, such as flip-flop, could offer valuable insights into algorithm effectiveness in diverse scenarios.

The End – Bibliography in README.txt