Randomized Optimization Report

Randomized Optimization is a method used in computer science to solve optimization problems by introducing randomness into the search process. It involves algorithms that make use of random variables to find solutions more efficiently. These methods are useful for solving optimization problems where traditional algorithms may struggle to find the optimal solution.

Interesting Optimization Problems:

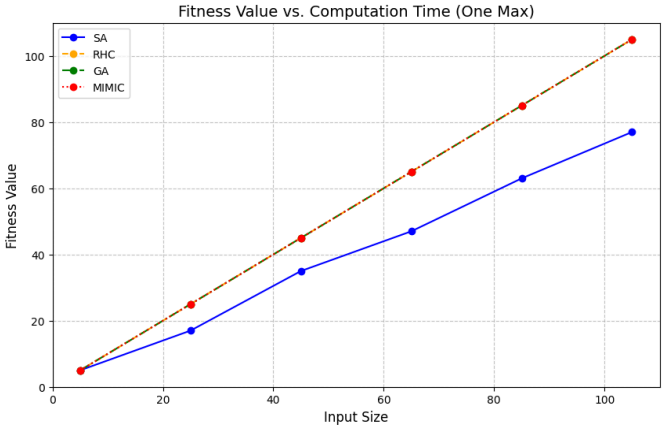
One Max Optimization Problem: The One Max problem is a classic optimization problem used in the field of computation. In this problem, the goal is to find a binary string of fixed length that consists of all 1s. The objective is to maximize the number of 1s in the string. Mathematically, it can be described as maximizing the sum of the elements in the binary string. The maximum fitness (optimal solution) is achieved when all bits are set to 1. The One Max problem is straightforward to understand. Hence it can serve as an introductory problem in understanding optimization algorithms like RHC, SA, GA, and MIMIC. The problem scales very well, making it perfect for understanding optimization algorithms and their performance on different problem sizes (input size of bit string). The One Max problem lacks local structure, and the fitness landscape is relatively flat. This can make it less difficult for optimization algorithms that rely on exploiting local information. As the optimization progresses, individuals in the population converge toward the optimal solution. This easy terrain of the problem makes the algorithm find global optima (only one present) and thus is easy to optimize. Algorithms that perform well on the One Max problem may not necessarily generalize to more complex problems because there is no plateau or multiple local optima present which becomes difficult to optimize. Thus for all these reasons One Max optimization problem is a good starting point for understanding how optimizations work under the hood with all hyperparameter tuning etc.

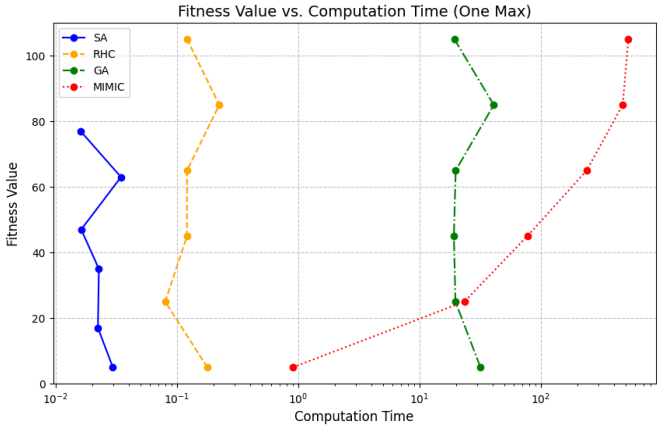
Continuous Peaks Optimization Problem: The Continuous Peaks Problem is defined on a binary string of length **n**, where **n** is typically large in our problem, it is 105 at max, and the goal is to maximize the number of consecutive bits that are either all 0s or all 1s. The problem is designed to be challenging for optimization algorithms because it has an ambiguous nature structure. The fitness of a binary string is determined by the maximum number of consecutive bits that are either all 0s or all 1s. The goal is to find a binary string with the highest fitness value. The fitness landscape of the Continuous Peaks problem is defined by a flat region of high fitness in the middle surrounded by regions of lower fitness. This structure makes it difficult for optimization algorithms, as it may mislead them into converging early to a suboptimal solution. The problem also contains local optima, where a small change to the bit string results in a decrease in fitness. These local optima can mislead search algorithms, making it challenging to find the global optimum. The ambiguous aspect of the problem arises from the fact that improving the fitness of the string within the flat region might not necessarily lead the algorithm toward the global optimum. This makes it challenging for optimization methods. On top of these choosing a threshold value of 0.35 makes the problem even more herculean because the threshold value is used to filter out plateau and small peaks, allowing the algorithm to focus on identifying larger and more important peaks. Here I selected 0.35 as the threshold value but also experimented with a threshold to see how it behaves in a difficult terrain of Continuous peaks. After all these challenges Continuous Peak became an interesting problem that can be applied to the context of randomized optimization algorithms, such as GA, SA, MIMIC, and RHC. It provides a scenario where these algorithms can be tested for their ability to handle ambiguous landscapes and avoid convergence to noisy false peak solutions.

Four Peaks Optimization Problem: The 4-Peak Optimization Problem is a combinatorial optimization problem often used to test optimization algorithms [1]. It is an ambiguous problem that exhibits both global and local optima (plateau), making it difficult for optimization algorithms to find the global optimum. In the 4-Peak problem, a bit string of length **n** is considered, where **n** is typically large in our problem, it is 105 at max. The bit string represents a state space, and each bit corresponds to a decision variable. The goal is to find the configuration of bits that maximize a fitness function. The 4-peak problem has the flexibility to find peaks with a minimum threshold. This allows more probability of a combination of heights that need to be found for optimal solutions. An additional point is allocated if we find a solution beyond the threshold. This means the problem keeps finding a solution until it gets the minimum high for the peak as/beyond the threshold value. Here I selected 0.45 as the threshold value but also

experimented with a threshold to see how it behaves in a difficult terrain of 4 peaks. The 4-peak problem is more scalable as it can be applied to more complex problems with more variables. The 4-peak problem is computationally more complex as there are more variables to optimize. This means that the algorithm may take longer to run and may require more computational resources. With four peaks, there are more local optima that the algorithm can get stuck in. This means that the algorithm may have a harder time finding the global optimum, requiring more iterations to converge. The 4-Peak problem is challenging for optimization algorithms due to ambiguous plateaus. Ambiguous plateaus make it difficult for algorithms to distinguish between good and bad solutions, leading to slow convergence (RHC) and exploration of the search space. All these issues help us understand different optimization solutions like RHC, SA, GA, and MIMIC and for all these reasons 4-peak is a great choice.

One Max Optimization Problem: The One Max problem as defined above has to maximize the number of 1s in the string where the initial expectation is that RHC will perform well because there is only one global optima and RHC is good at finding the same. Simulated Annealing will also converge fast to the maximum fitness value because high temperature and exponential Decay hyperparameters will help maximize the 1s in the string. The genetic algorithm will perform the best, The reason for this is the population and mutation increase the probability of getting the perfect fitness value. Mimic could perform similarly to other optimization algorithms because of its structural design but since mimic is very computationally heavy it might take a long time to perform and get the best fitness value.



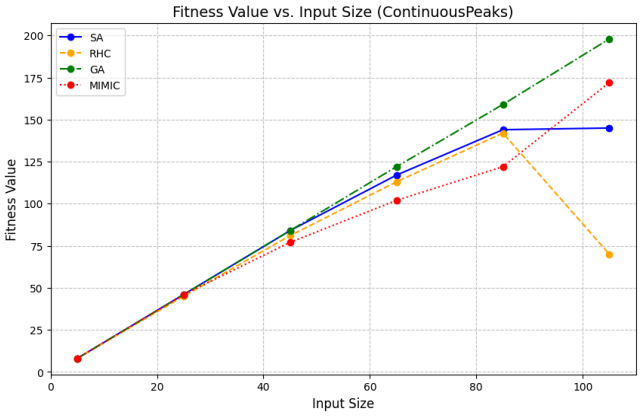


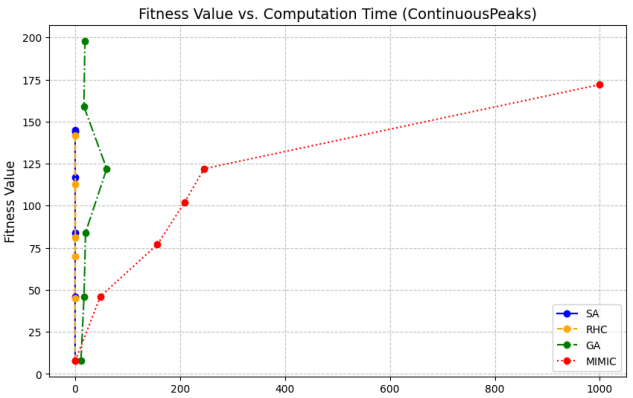
The performance of RHC, GA, and MIMIC is similar but the performance of SA is poor on a simple algorithm like One Max Optimization. The RHC, GA, and MIMIC have been optimized with the best hyperparameter. SA has also been applied with a hyperparameter of init_temp of 100 but it seems inefficient in performance as SA is unable to converge to a better solution. When tried init_temp of 500 individually on the One Max problem with SA optimization it generated a straight line as we are getting for other algorithms. RHC, GA, and MIMIC performed similarly, and as per expectations hyperparameters like a restart for RHC, mutation_prob, and pop_size for GA, and keep_pct and pop_size have been set to maximum values to achieve perfect fitness value and this can be observed in Fitness value Vs. Computational Time graph too.

For a bit-string of length 50, I compared the fitness over computation time. MIMIC reaches the highest fitness value but has the longest computation time as per expectations. RHC and GA attain a moderately high fitness value with a shorter computation time compared to MIMIC but GA takes longer due to its computational mutation probability and population size set to 0.5 and 500 respectively. SA lands on a relatively lower fitness value but has the fastest computation time. MIMIC might be suitable if achieving the absolute best solution is crucial, even if it takes longer. RHC could be a good compromise between solution quality and efficiency. GA might be a balanced option if fitness is important but has extra computational power. SA could be preferable if speed is the primary concern, although the solution quality might be sacrificed if not optimized for hyperparameters.

Continuous Peaks Optimization Problem: The Continuous Peaks Problem has the goal of maximizing the number of consecutive bits that are either all 0s or all 1s. Since the continuous peaks are quite a difficult problem so initial expectation with RHC is low because of the complexity of peak and plateau while finding consecutive 1/0

will be difficult for RHC. SA might perform okay provided hyperparameter tuning is done to understand the structure of the peaks considering high temperature and other parameters like exponential decay. GA can do well because multiple factors of population and mutation make it a favorable algorithm for escaping the plateau and reaching the optimal peak. MIMIC should perform the best because its structure is best suited for continuous peak kinds of issues. MIMIC might consume more time since it is a computationally heavy algorithm.

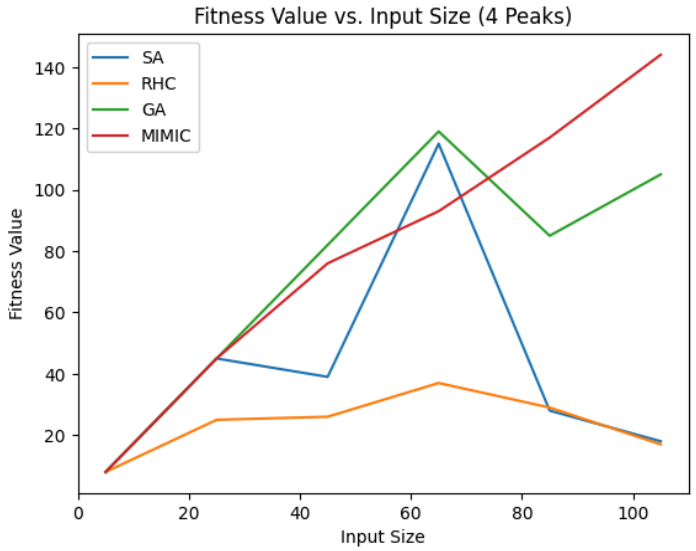


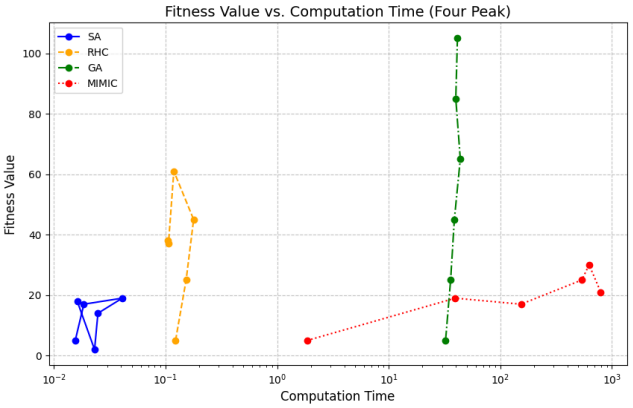


All the algorithms have similar performance for an input size of 40. Once that input size is surpassed we get to see robust GA optimization performing the best as per expectations. The reason for GA's great fitness value is the hyperparameter mutation_prob as 0.5 and pop_size as 500 is perfect as per computational power and is a perfect fit to achieve a high fitness value for a given input size (maximum 105-bit string). SA did perform well in continuous peak optimization reason for this is SA has been tuned for hyperparameters with a temperature of 500 and used exponential decay to boost performance. The poor performance in the One max problem led to this ideal value which became a good performance for continuous peak. RHC had good fitness value for initial values but as input size increased the fitness dropped drastically and the reason for the same explains the simplicity of the algorithm which can't handle the complex plateau and peak terrain. Technically the hyperparameter has been tuned to restart value as 10 which has been worked initially but it doesn't hold for large input sizes. MIMIC surprisingly performed poorly though it took most of the time for execution which was expected but the performance should have been good. Hyperparameters keep_pct and pop_size have been set to the best values and the performance degrades over time.

For a bit-string of length 50, I compared the fitness over computation time. Where RHC and SA attain a moderate fitness value with a shorter computation time. GA achieves a good fitness value with a computation time between SA and MIMIC. MIMIC lands on a relatively higher fitness value as compared to SA and RHC but has the slowest computation time. MIMIC could be a good compromise between solution quality and efficiency with speed and computational power not being a concern. GA might be a balanced option if both fitness and computation time are important. SA or RHC could be preferable if speed is the primary concern, although the solution quality might be sacrificed.

Four Peaks Optimization Problem: The 4-peak problem as defined above is an ambiguous problem where initial expectations are RHC might perform well for smaller values but will perform worse later because it is too trivial to handle plateau. Simulated Annealing is expected to perform okay at the start but should increase fitness value post-adjusting to perturbation and temperature values. The genetic algorithm will perform well since it has the strength of hyperparameters like mutations and populations which help in gliding through a plateau and reaching global optima. The MIMIC is the flagship optimization algorithm and it is expected to perform the best on the 4-peak reason for this is the tree-like structure to reach optimal value and bypassing the local optima makes it robust to performance.





For a small input size of a 4-peak problem, all algorithms performed the same in terms of fitness value because the chances of local optima are less as compared to global optima. As the input size increases the behaviour of different algorithms is very interesting. GA performed very well in the range of 25-70 input size with maximum as the fitness value and the reason behind this is the hyperparameter pop_size set to 500 and mutation_prob as 0.5 because of these values the probability of mutation is 50% in large population size which increases the performance of the optimization. RHC performed poorly despite tunning for restart hyperparameter because once it enters the plateau of 4-peak it gets stuck with local optima and fitness value decrease for RHC. The SA did perform well with mid-sized input values but there is a large drop in the fitness this is because the hyperparameter with exponential Decay rate performed well with an input size of 60 but was unable to perform as size increases. The algorithm may need more time to converge to the global optimum. MIMIC optimization had a great start in the beginning while its fitness score went low with the mid-input sized string but performed the best as hyperparameter pop_size=500 and keep_pct=0.5 learned the problem space well and reached global optima.

Hyperparameter tuning is required of RHC, when the restart value is 10 and max_iters is 1000 the performance of RHC increases drastically because A higher restart value allows the algorithm to explore a larger search space, while a lower value focuses more on exploiting the current best solution. In the case of iterations, a higher value allows the algorithm to perform more iterations and potentially find a better solution, while a lower value will stop the algorithm earlier, potentially missing out on a better solution. This is what happened in the previous problem and then tuning led to good results. For SA hyperparameter tuning, when the exponential decay rate with init_temp as 500 was set the algorithm performed decently as the high temperature cools down over time, allowing the algorithm to converge to a better solution.

For a bit-string of length 50, I compared the fitness over computation time. Where GA reaches the highest fitness value (around 110) on the y-axis. It achieves this high fitness value with a moderate computation time (around 10 on the x-axis). RHC has a faster computation time than SA (around 0.1), it achieves a lower fitness value (around 60). SA has a slightly lower computation time than RHC but also reaches a lower fitness value than RHC (around 20). MIMIC: This algorithm has the slowest computation time (around 100) but also has the lowest fitness value (around 25). If achieving the absolute best solution is important, and computation time is not a major constraint, then GA seems like a good choice based on this graph. If you need a faster solution and are willing to sacrifice some fitness value, then RHC or SA might be options to consider. For a more balanced approach between fitness and computation time, and can explore parameter tuning for MIMIC to see if its performance can be improved.

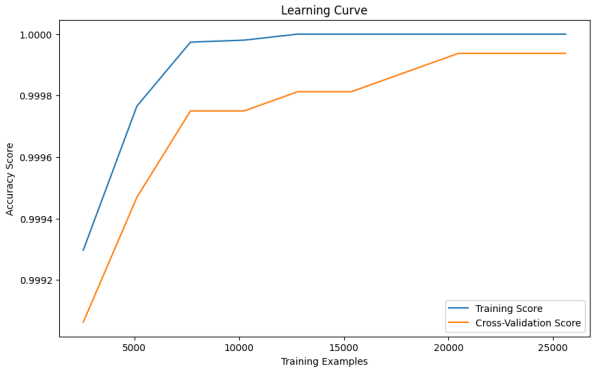
One Max Optimization vs Four Peaks Optimization vs Continuous Peaks Optimization:
One Max provides a simple starting point, while Four Peaks introduces deception, and Continuous Peaks adds a flat fitness region. This gradation allows for a systematic analysis of algorithmic performance as problems become more complex.

RHC and SA primarily depend on local search, while GA and MIMIC balance searching diverse solutions and exploitation of the terrain for the bit string problems. RHC and SA generally have faster computation times, while GA and MIMIC can be slower, especially for complex problems hence the algorithm is strictly according to the computational power. RHC and SA are prone to get stuck in local optima, while GA and MIMIC have mechanisms (mutation, population diversity, and decision tree structure) to overcome these challenges. All algorithms require careful hyperparameter tuning for optimal performance, but the degree of change and tuning might vary. The best choice of algorithm depends on the specific problem characteristics and optimization goals. If achieving the absolute best solution is crucial and computation time is not a major concern, GA or MIMIC might be preferable. If speed is essential and some compromise in solution quality is acceptable, RHC or SA could be considered.

Conclusion: GA consistently performs well across different problems, especially with suitable hyperparameters. SA and RHC are faster but may sacrifice solution quality. They can be preferable for speed-oriented scenarios. MIMIC is a good compromise between solution quality and efficiency but can be computationally heavy.
GA had the best performance because of its ability to explore the search space effectively. The algorithm works by generating a population of candidate solutions, evaluating their fitness, and then iteratively improving the population through crossover, mutation, and selection. The mutation probability has the power to select the population percentage for mutation each generation. In our case mutation probability was set to 0.5 which means every 50% population has the chance to mutate. Once an individual is selected for mutation, the algorithm applies a mutation operator to the individual's genes. Combine this with a pop_size of 500 which created the robust optimization algorithm that has the ability to glide through the local optimas and find the best global optimum solution in bit string problems. Here if we consider time vs GA performance then it is slower than RHC or SA per se but faster than MIMIC. With more hyperparameter tuning we can achieve state-of-the-art fitness value with GA.

Neural Networks:
The Dataset used in Assignment 1 was regarding Memory-based Malware Artifacts. The main objective behind choosing a dataset was pre-labeled data. This allows the algorithm to categorize data as either benign or malware. While the features were diverse, the model faced challenges due to imbalanced data (much more benign data than malware). To handle imbalance data had to use LabelEncoder[2]. The dataset contained 40934 Malware/Benign data points which have to be classified by the NN. 55 features in the dataset are numerical in type and hence no additional work of encoding is required here. Earlier in Assignment 1, I applied a hidden size of 128, and learning rate of .001, and an activation function as relu which yielded results with an accuracy of 99.9875%. Hence considering these as ideal parameters because of hyperparameter tuning and cross-validation already applied in Assignment 1 and I ran back_prop and other optimization algorithms. I will use both the maximum number of iterations, which is set to 4000, and the other parameters for this study. Also, used Adam as an optimizer to apply adaptive learning rate, momentum, and weight decay on NN for better performance.

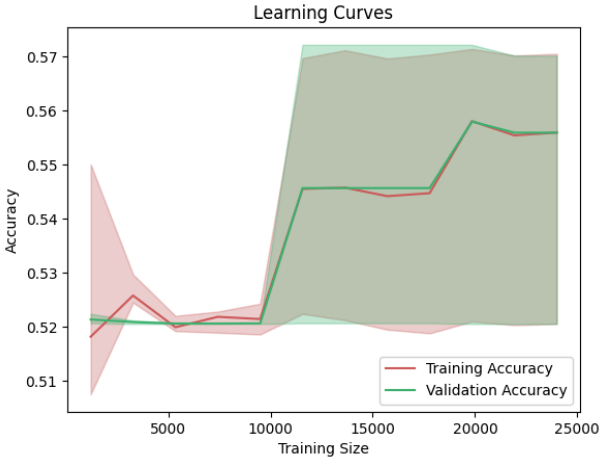
The results of the neural network using the Adam solver and a one-layer structure with 100 nodes are shown below.



MIrose_hiive Backprop algorithm with gradient descent as an optimizer and early stopping to prevent overfitting, resulted in an accuracy of 55%. The algorithm was able to effectively learn the underlying patterns in the data, and accurately classify instances with a moderate level of confidence. To further improve the performance of the

algorithm, the addition of exponential decay to the early stopping mechanism can be implemented. This technique involves gradually reducing the learning rate of the optimizer as the training progresses, which helps to prevent overfitting by slowing down the rate at which new information is incorporated into the model

Train Accuracy: 55.2968
Test Accuracy: 55.3254



When the hidden size is 128 neurons and all other hyperparameters are set to optimal level then we get an accuracy of about 55%. Backprop allowed gradient descent to explore the neural network terrain and find optimal weights. MLrose_hiive Backprop will act as a benchmark for comparing other optimization algorithms.

The Random Hill Climb algorithm was applied to a Neural Network with a hyperparameter restart set to 10, resulting in a disappointing accuracy score. The tuning process for max_attempt and learning rate was brute forced, with a maximum attempt count of 1000 and a learning rate of 0.01. The RHC algorithm cannot replace the Backprop algorithm in finding the optimal weights and biases for a Neural Network. The reason for this is that the RHC algorithm is a random search algorithm, meaning that it relies on randomness to explore the search space, but the Backprop algorithm uses a more guided search strategy based on gradient descent. As a result, the Backprop algorithm is more effective at finding the optimal solution, especially when the search space is large and complex dataset Memory-based Malware Artifacts. The RHC algorithm has some advantages over the Backprop algorithm. For example, it is faster to compute than the Backprop algorithm, when the absolute best solution is not important. While the RHC algorithm may not be as effective as the Backprop algorithm in finding the optimal weights and biases for a Neural Network, it can still be a useful tool in certain situations. Its faster computation time makes it a valuable alternative to the Backprop algorithm, especially when the absolute best solution is not necessary.

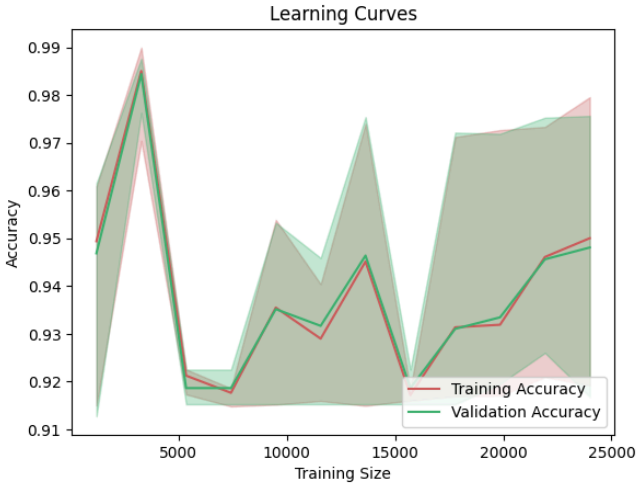
Train Accuracy: 47.9093
Test Accuracy: 47.9625

The performance of the MLrose_hiive Neural Network optimized with simulated annealing for weights and biases was found to be similar in accuracy compared to the Random Hill Climb algorithm. This was confirmed through double checking and the reason for this behavior is that both RHC and SA can have similar performance on some optimization problems, particularly when the search space is complex and has many local optima in dataset Memory-based Malware Artifacts. Furthermore, it was also observed that even tuning the hyperparameters did not yield different results. Init_temp has been initialized with different values but the optimal was 500. This suggests that both algorithms are capable of exploring the search space and finding good solutions. SA is a stochastic optimization algorithm that uses a probabilistic approach to find the search space and adapt to the constraints. We can say RHC and SA has almost similar performance when NN backprop is replaced with an optimization algorithm.

Train Accuracy: 47.9547
Test Accuracy: 47.892

The Genetic Algorithm has the best performance in optimizing the Neural Network (NN). The GA was able to achieve an accuracy score of 98%. GA is too sensitive for training size and can overfit very easily hence we chose early stopping to avoid overfitting. The GA was able to avoid overfitting by using early stopping, which involves monitoring the validation loss during training and stopping the training process before the model overfits the data. Another reason for the GA's success is the optimization of hyperparameters. The GA was able to optimize the mutation probability and population size hyperparameters. The GA was also able to optimize the learning rate hyperparameter. The GA was able to optimize the max_attempt hyperparameter. By optimizing this hyperparameter, the GA was able to find the optimal solution more efficiently and avoid getting stuck in local optima. Hyperparameter tuning with max_attempt lower to 50 and learning rate set to 0.01 because of computational limitation we achieved optimal results. If the learning rate was set to 0.001 then the learning curve was disappointed because the NN was unable to learn with such small rates and accuracy went to 85%. Tuning it to 0.01, the decision was taken the get high accuracy. Also, there was not much difference when max_attempt was lowered from 1000 in the case of RHC/SA to 50 for GA. This decision saved a lot of computational power. The GA's ability to optimize the mutation probability, population size, learning rate, and max_attempt hyperparameters was particularly effective in achieving the best possible performance.

Train Accuracy: 98.771875
Test Accuracy: 98.7625



Conclusion:

Algorithms	Train Accuracy	Test Accuracy
Backprop	55.2968	55.3254
RHC	47.9093	47.9625
SA	47.9547	47.892
GA	**98.771875**	**98.7625**

All these results are specific to dataset Memory-based Malware Artifacts and results may vary based on the problem chosen. The performance of the GA on the dataset is promising, and it demonstrates the potential of replacing the Backpropagation algorithm with the GA optimization algorithm for initializing weights and biases. Although the GA requires time to converge to accurate values, it has consistently performed the best across different algorithms. In contrast, the RHC and SA algorithms were observed to get stuck in local optima, resulting in a decline in accuracy which can be observed from the above table. The GA's ability to efficiently explore the search space and avoid local optima is an advantage, making it a suitable choice for optimizing neural networks.

References

[1]. https://en.wikipedia.org/wiki/Combinatorial_optimization.
[2]. Supervised Learning Report. Assignment 1