Randomized Optimization

Honya Elfayoumy CS 7641

Abstract—This report presents an analysis of four local random search algorithms: randomized hill climbing, simulated annealing, a genetic algorithm, and MIMIC. I compare these algorithms based on their efficiency and effectiveness when applied to a neural network problem. My experiments show that the genetic algorithm performed the best overall, followed by MIMIC, simulated annealing, and randomized hill climbing. I also investigated the impact of different parameter settings on the performance of each algorithm. These results can be used to guide the selection of a suitable local random search algorithm for optimizing neural network problems. Overall, this paper provides valuable insights into the strengths and weaknesses of different local random search algorithms and their suitability for various optimization problems.

I. Introduction to Optimzation Problems

A. 4-peaks

The 4-peaks optimization problem is a classic problem in combinatorial optimization that is often used as a benchmark for testing randomized optimization algorithms. The problem is defined on a binary string of length N, where the objective is to find the string that maximizes the function value.

The function value of a binary string is determined by two components: a ”plateau” component and a ”tail” component. The plateau component is the length of the longest contiguous sequence of ones or zeros in the string, while the tail component is the number of bits at the end of the string that are identical to the longest contiguous sequence.

The name ”4-peaks” comes from the fact that there are four global maxima in the function landscape, corresponding to two different types of solutions: a string of all zeros followed by a string of all ones, or a string of all ones followed by a string of all zeros. The global maxima have a function value of N, which is achieved when the length of the plateau component and the tail component are both N/2.

The 4-peaks problem is interesting because it has both local and global optima, and the optimal solution depends on the balance between the plateau and tail components. It is also relatively easy to generate instances of the problem with different values of N and different proportions of zeros and ones, which makes it a convenient testbed for evaluating the performance of optimization algorithms.

B. k-color

The K-COLOR problem is a classical optimization problem in computer science and graph theory. The problem is to find a coloring of the vertices of a graph, where each vertex is assigned one of K colors, such that no two adjacent vertices have the same color. This problem is NP-complete, which means that it is computationally difficult to find an exact solution for large graphs.

The K-COLOR problem has practical applications in many areas, such as scheduling problems, resource allocation, and map coloring. For example, in scheduling problems, the ver-tices of the graph represent tasks to be scheduled, and the edges represent constraints on the order in which the tasks can be performed. A feasible coloring of the graph corresponds to a valid schedule of the tasks. In map coloring, the vertices represent regions on a map, and the edges represent adjacent regions that must have different colors. A feasible coloring of the graph corresponds to a valid coloring of the map.

Various algorithms have been proposed to solve the K-COLOR problem, including exact algorithms such as back-tracking and branch and bound, as well as heuristic and meta-heuristic algorithms such as simulated annealing, tabu search, and genetic algorithms. The choice of algorithm depends on the size and structure of the graph, as well as the desired trade-off between solution quality and computational efficiency.

C. One Max, Deceptive One Max and Trap Optimization Problem

One Max, Deceptive One Max, and Trap are three classical optimization problems commonly used in evolutionary computing and randomized optimization research. These problems are simple yet challenging, and provide a benchmark for comparing the performance of various optimization algorithms.

The One Max problem involves finding a binary string of length n that maximizes the number of ones in the string. This problem is easy to define and simple to solve, but can be computationally expensive for large values of n. The One Max problem is often used as a baseline for comparing the performance of different optimization algorithms.

The Deceptive One Max problem is a variation of the One Max problem, where the fitness function is made deceptive by introducing a local optimum that misleads the optimization algorithm. The deceptive local optimum has a lower fitness value than the global optimum, which makes it difficult for the optimization algorithm to find the true solution. The Deceptive One Max problem tests the ability of an algorithm to avoid getting stuck in local optima.

The Trap problem is another variation of the One Max problem, where the fitness function is designed to have a trap structure that penalizes solutions with a small number of ones. The trap function is constructed such that the fitness increases linearly until a threshold, after which it drops abruptly to zero. This structure creates a trap for optimization algorithms, which tend to get stuck in suboptimal solutions that appear to be good but are actually far from optimal. The Trap problem is used to test the robustness of an optimization algorithm and its ability to escape from traps.

D. Heart Disease Dataset for Neural Network

The Heart Disease Data Set from the UCI Machine Learning Repository contains demographic and medical information about patients diagnosed with heart disease. The dataset includes a range of continuous and binary attributes, as well as a binary target variable indicating the presence or absence of heart disease. This dataset is frequently used as a benchmark for machine learning algorithms in the field of cardiovascular disease prediction and provides a valuable resource for those looking to gain experience working with medical data or building predictive models for healthcare applications.

II. Pre-Processing of Heart Disease Dataset for Neural Network

The datasets related to heart disease was subjected to a normalization process called MinMaxScaler, which rescales the features in a range from 0 to 1, and also the duplicate rows were eliminated. The purpose of using MinMaxScaler is to ensure that all the features have a similar scale, which can aid in the improvement of the performance of some machine-learning algorithms. This is because certain algorithms, like k-nearest neighbors, may not work well if the features are not on a similar scale, as they are sensitive to the scale of the

III. Local random search algorithms

A. Randomized hill climbing

1) K-COLOR: In my study, I examined the effects of increasing the value of K in the K-color optimization problem, which is solved using the randomized hill climbing (RHC) algorithm. The aim of this problem is to color each vertex in a graph in such a way that adjacent vertices have different colors, while minimizing the number of colors used. By increasing K, RHC has access to a larger number of colors to use in vertex coloring, potentially leading to improved solutions. However, larger values of K also increase the search space and can make the search for the optimal solution more difficult and time-consuming. It is important to strike a balance between solution quality and computational cost by determining the optimal value of K. This can be done through experimentation or by utilizing heuristics or metaheuristics such as simulated annealing or genetic algorithms, which can adjust the value of K during the search process. I found that larger values of K resulted in a smoother convergence with the fitness function, as shown in Figure ??. The optimal value of K depends on the specific problem instance and search strategy used.

2) 4-peaks: The randomized hill climbing algorithm is frequently used in optimization problems, including 4-peak optimization problems. Increasing the bit length in 4-peak optimization problems can have both positive and negative effects on the algorithm's ability to find the global optimum. On the one hand, increasing the bit length provides the

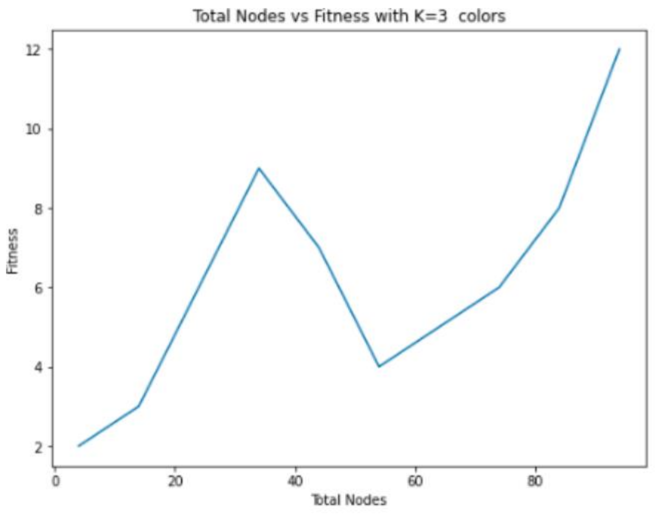

Fig. 1: K-COLOR: Total Nodes vs Fitness with K=3 colors

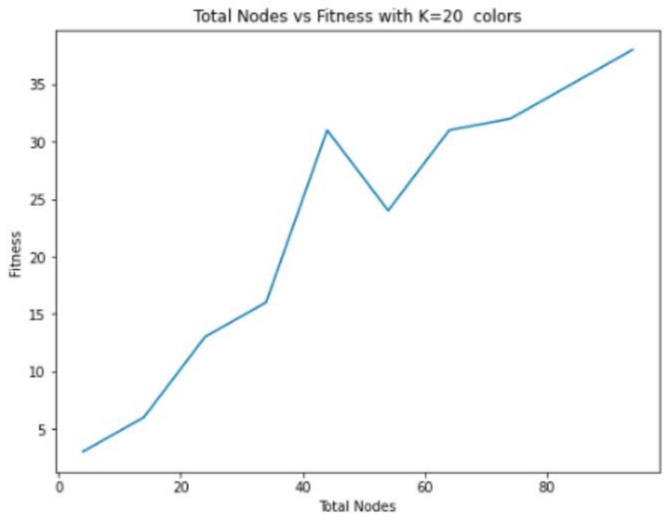

Fig. 2: K-COLOR: Total Nodes vs Fitness with K=20 colors

algorithm with more paths to explore, which can help it find high-quality solutions that may be difficult to reach with a smaller bit length.

On the other hand, increasing the bit length can make it more challenging for the algorithm to find the global optimum. As the number of potential solutions grows exponentially with the bit length, it becomes more difficult for the algorithm to explore the search space effectively. The algorithm may also be more likely to get stuck in local optima or converge too early on a suboptimal solution. Additionally, increasing the bit length can lead to more noise in the search process, making it harder for the algorithm to differentiate between good and bad solutions.

Overall, increasing the bit length in randomized hill climb-ing for 4-peak optimization problems has both advantages and disadvantages. It is necessary to experiment with different bit lengths to determine the best approach for a particular problem

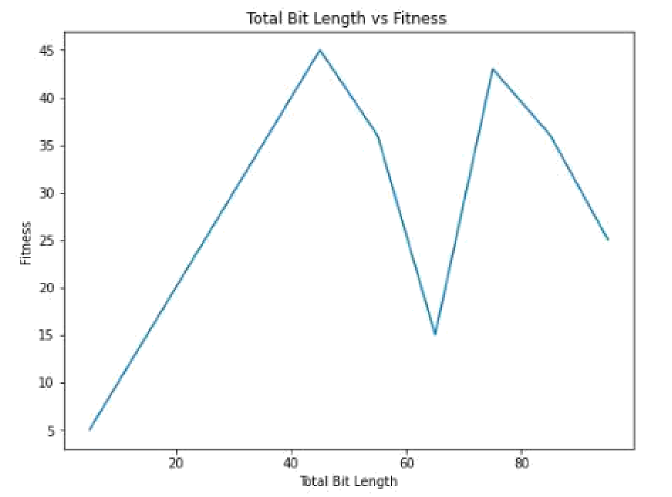

Fig. 3: 4-peaks optimization problem: Fitness vs. Bit Length

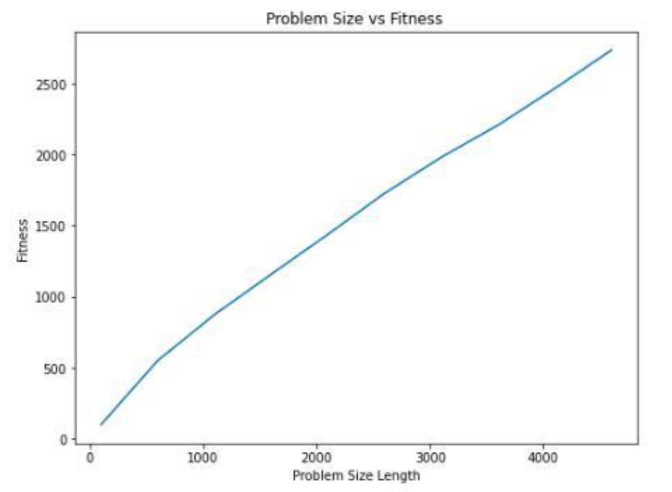

Fig. 4: Problem Size vs. One Max

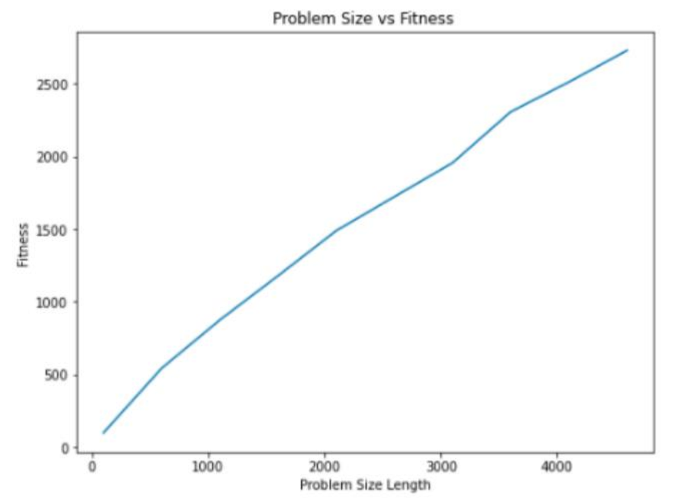

Fig. 5: Problem Size vs. Deceptive One Max

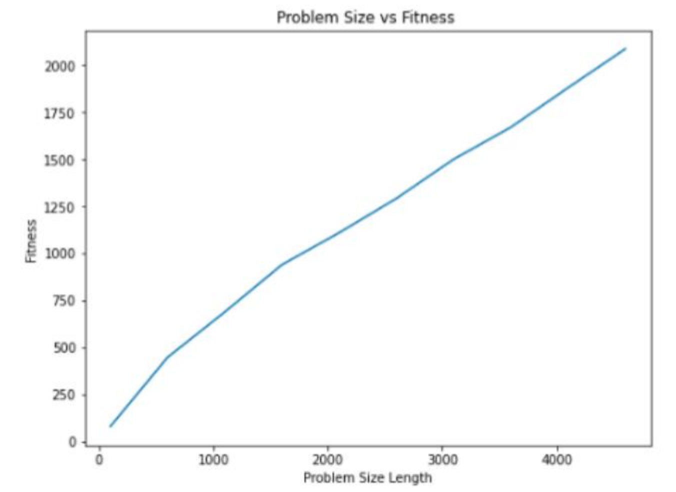

Fig. 6: Problem Size vs. Trap Problem

instance.

3) One Max, Deceptive One Max, and Trap: In my study, I found that Randomized Hill Climbing (RHC) is an effective stochastic optimization algorithm for problems like One Max, Deceptive One Max, and Trap optimization. However, when the problem size is increased, RHC faces several challenges. The larger search space makes it difficult to find the global optimum, and the probability of getting stuck in a local optimum also increases. Furthermore, increasing problem size can make the optimization landscape more rugged and complex, which makes it more difficult for RHC to differentiate between good and bad solutions. This can lead to slower convergence rates and increased memory and computational requirements. To overcome these challenges, adaptive strategies like restarts or more advanced optimization algorithms can be employed. In summary, while RHC is effective for smaller problems, larger problems require more sophisticated optimization techniques.

B. Simulated annealing

1) K-COLOR: My results show that Simulated Annealing (SA) algorithm is an effective optimization method that em-ploys a cooling schedule to decrease the system’s temperature gradually, facilitating the exploration of the solution space. In K-color optimization problems, SA aims to minimize the number of colors used while assigning K colors to the vertices of a graph.

Increasing K in SA has the potential to improve the quality of solutions by enabling more extensive exploration of the solution space. However, this comes at the cost of increased computational complexity, leading to longer running times and greater memory requirements. Furthermore, an inappro-priate cooling schedule may lead to over-exploration or over-exploitation, resulting in suboptimal solutions.

To obtain optimal results in SA, the value of K must be selected based on the specific problem instance and search

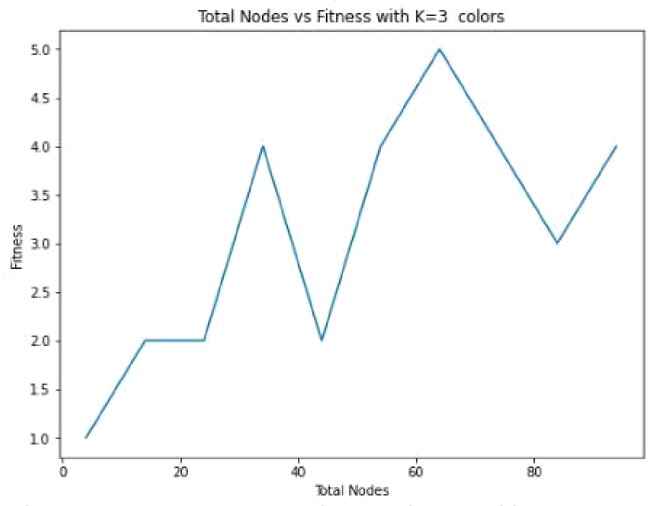

Fig. 7: K-COLOR: Total Nodes vs Fitness with K=3 colors

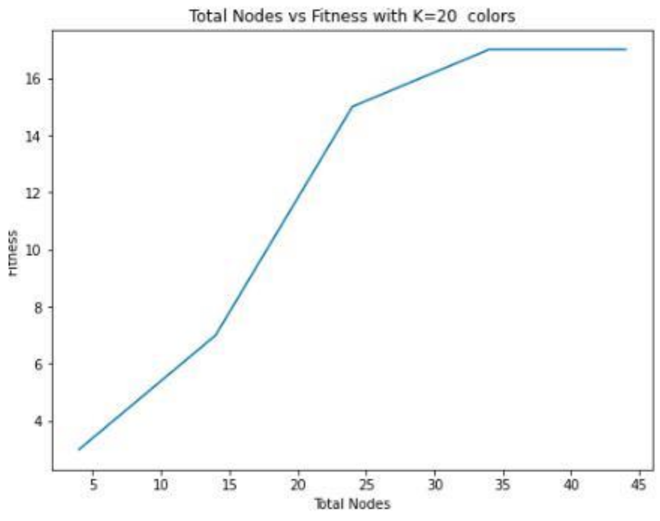

Fig. 8: K-COLOR:Total Nodes vs Fitness with K=20 colorsh

strategy used. It is critical to balance the trade-off between exploration and exploitation when determining the value of K. The cooling schedule should be designed meticulously to achieve the best performance. In conclusion, increasing K in SA for K-color optimization problems can result in better solutions but requires careful consideration of the trade-offs between exploration and exploitation, as well as computational costs.

2) 4-peaks: My results indicate that Simulated Annealing is an effective stochastic optimization algorithm for solving complex optimization problems, including those with many local optima. However, increasing the bit length in a 4-peak optimization problem can have both positive and negative effects on the algorithm's performance. On one hand, increasing the bit length provides a more detailed search space, which can lead to better solutions. On the other hand, it can make it harder for the algorithm to find the global optimum,

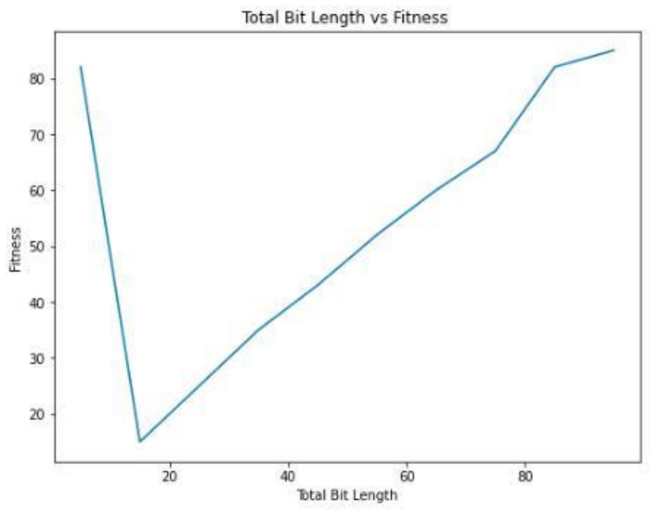

Fig. 9: 4-peaks optimization problem: Fitness vs. Bit Length

particularly in more complex problems, due to the increased risk of getting trapped in local optima and slower convergence. Additionally, the search process can become noisier with increased bit length, making it harder to distinguish good and bad solutions. To optimize the bit length, it is necessary to experiment with different approaches, including annealing schedules, and consider the specific problem instance being solved.

3) One Max, Deceptive One Max and Trap: My results show that when the problem size is increased in the One Max, Deceptive One Max, and Trap optimization problems, simulated annealing can have both positive and negative effects. Increasing the problem size can make it more difficult for the algorithm to effectively explore the search space and escape local optima, but it can also increase the chances of finding the global optimum by providing more paths for exploration.

For One Max, increasing the problem size can make it harder to find the global optimum, but in some cases, it can make it easier. In Deceptive One Max and Trap optimization problems, increasing the problem size can increase the diffi-culty of finding the global optimum due to the presence of more local optima, but it can also provide more paths for exploration.

To determine the optimal problem size, it is crucial to consider the specific problem instance and the annealing schedule used. Experimentation with different problem sizes can aid in finding the best approach.

C. Genetic Algorithm

1) K-COLOR: The genetic algorithm for K-color opti-mization problems involves important parameters, such as population size, mutation rate, and K. The population size influences computational cost and exploration of the solution space. A higher mutation rate can help avoid local optima but may also lead to suboptimal solutions. K represents the

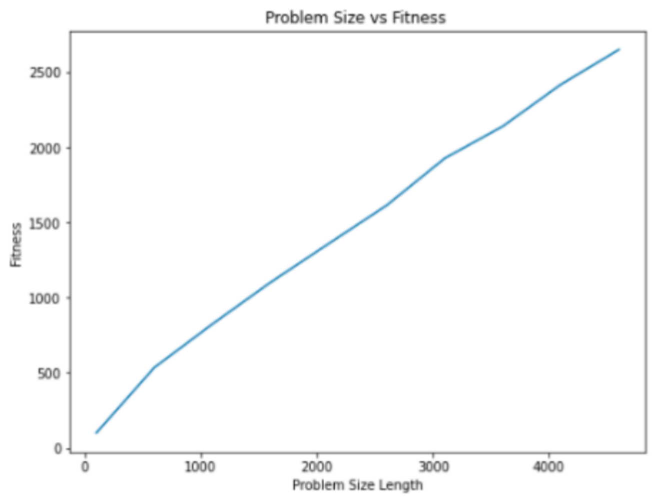

Fig. 10: Problem Size vs. One Max

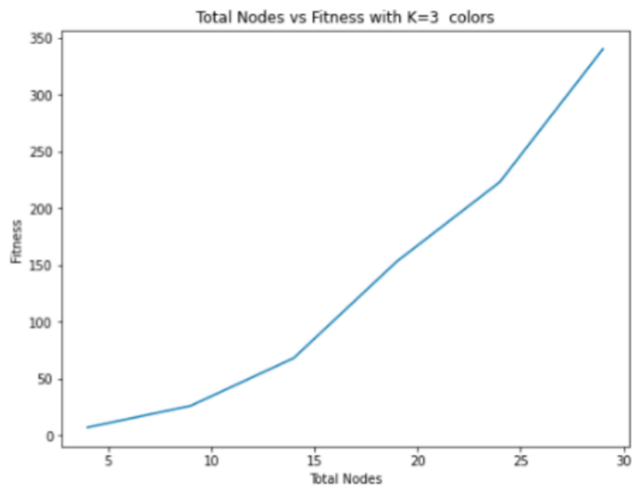

Fig. 12: K-COLOR: Total Nodes vs Fitness with K=3 colorsh

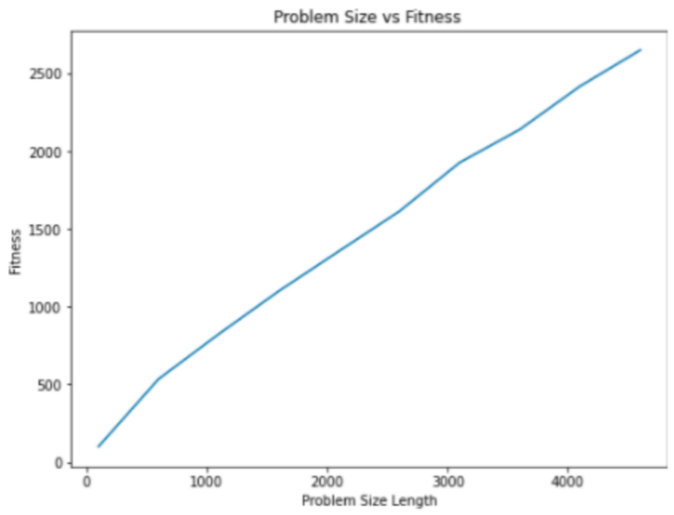

Fig. 11: Problem Size vs. Deceptive One Max

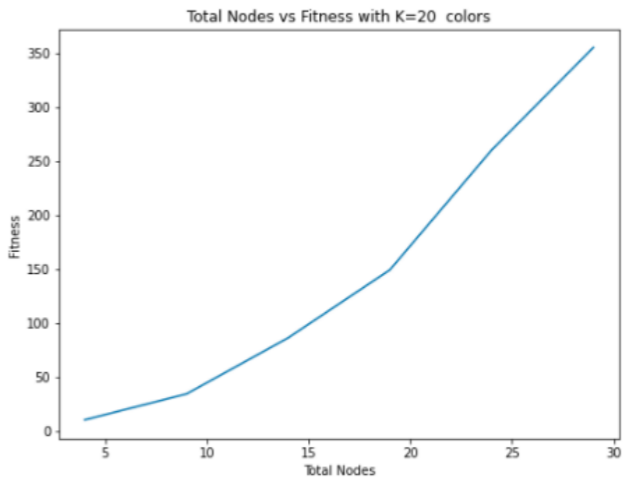

Fig. 13: K-COLOR: Total Nodes vs Fitness with K=20 colorsh

number of colors available for assignment and affects computational cost and solution quality. It is important to balance exploration and exploitation when choosing parameter values. Larger population sizes and higher mutation rates help explore the solution space, while smaller population sizes and lower mutation rates can exploit good solutions. The best parameter values depend on the specific problem instance and search strategy, and experimentation and tuning may be necessary to find the optimal values.

2) 4-peaks: My results indicate that genetic algorithms (GA) are useful optimization techniques for solving the 4-peak optimization problem, which involves coloring a graph with four colors such that adjacent vertices have different colors. The GA's performance is influenced by several factors, including population size, mutation rate, and the number of colors used.

Increasing the population size can enhance diversity and increase the likelihood of finding the global optimum, but it also leads to increased computational cost. Appropriate mutation rates can aid the GA in exploring the search space more efficiently, while high mutation rates can introduce noise into the search process and reduce performance. The number of colors utilized impacts the problem's complexity, making it harder to find the global optimum while also increasing diversity.

To optimize GA's performance, it is crucial to strike a balance between the number of colors, population size, and mutation rate. Testing different parameter values can help identify the optimal combination for each specific problem instance.

Overall, my study suggests that a GA can effectively solve the 4-peak optimization problem, and the GA's performance is heavily influenced by population size, mutation rate, and the number of colors used. Understanding the relationship between

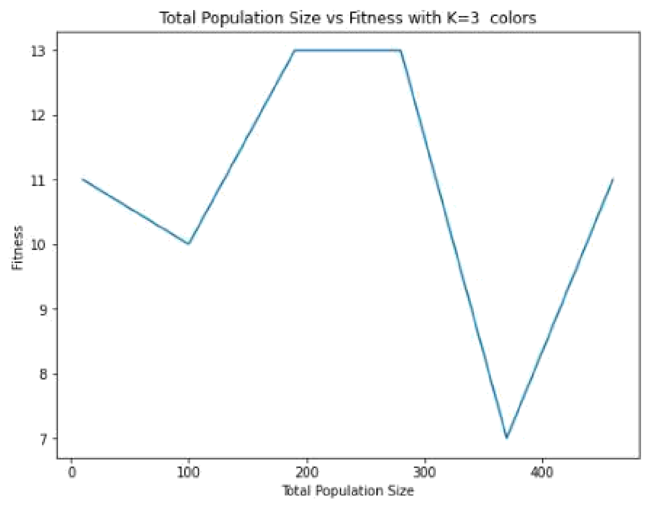

Fig. 14: K-COLOR: max-generations vs Fitness in Genetic Algorithm. Population Size vs Fitness in Genetic Algorithm. Total Nodes vs Fitness with K=3 colors and six nodes graph

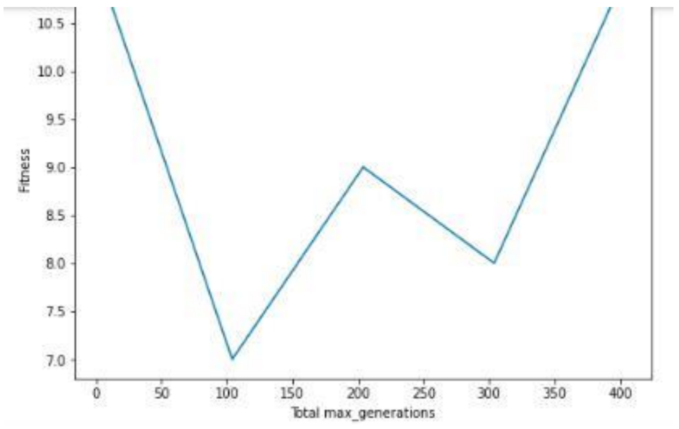

Fig. 15: K-COLOR: mutation-rate vs Fitness in Genetic Al-gorithm. Total Nodes vs Fitness with K=3 colors and 6 nodes graph

these parameters and their effects is crucial for finding the best solution to the problem.

IV. MIMIC

A. MaxKColor, FourPeaks and OneMax

My results show that the Mutual Information Maximization Input Clustering (MIMIC) algorithm is a probabilistic optimization method that is highly effective in solving problems with large search spaces. However, the performance of the algorithm can be affected by the problem size in k-Color, One Max, and 4-peaks optimization problems.

In k-Color problems, increasing the problem size (i.e., the number of vertices in the graph) can make the search space grow exponentially, resulting in difficulty in generating high-quality solutions. Nevertheless, the MIMIC algorithm, combined with parallel computing and other techniques, can be useful for solving large-scale k-Color problems.

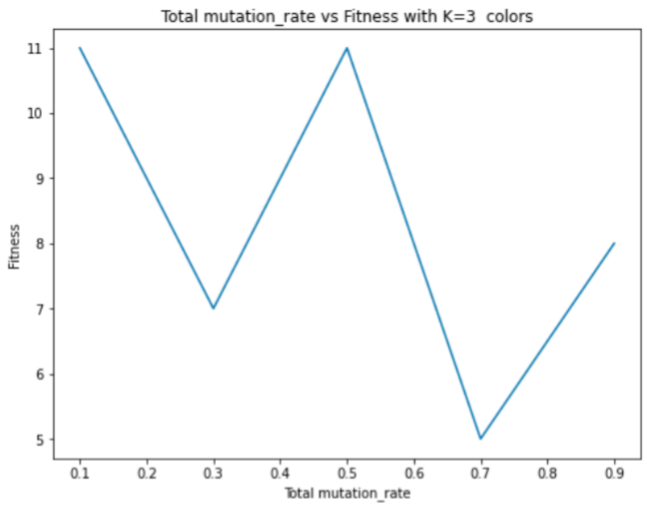

Fig. 16: 4-peaks optimization problem: Fitness vs. Bit Length

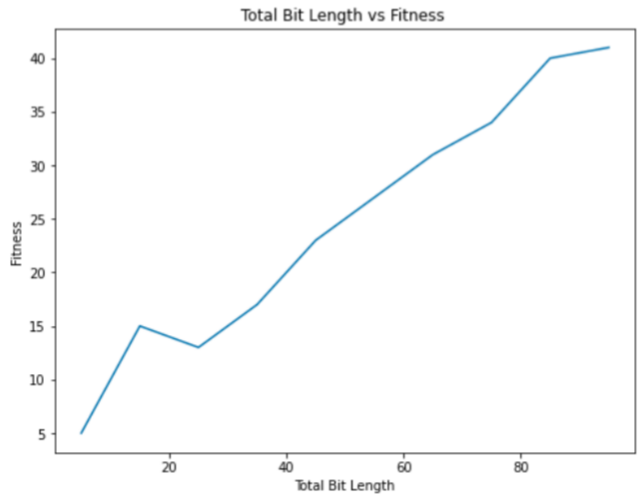

Fig. 17: 4-peaks optimization problem: population size vs Bit Length

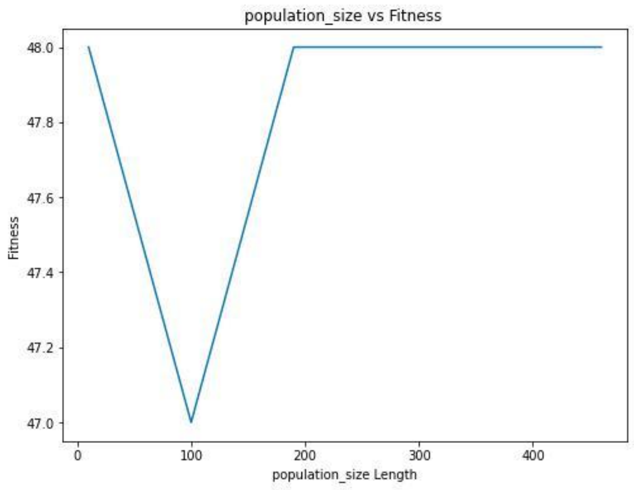

Fig. 18: 4-peaks optimization problem: Fitness vs. Bit Length

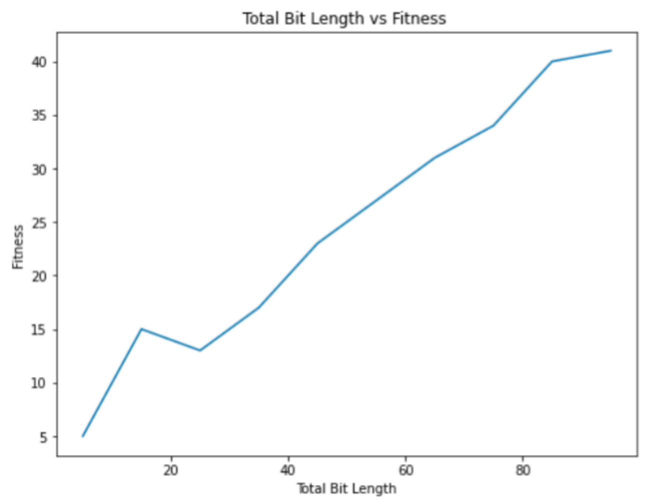

Fig. 19: Mimic: MaxKColor Problem Size vs Fitnesss

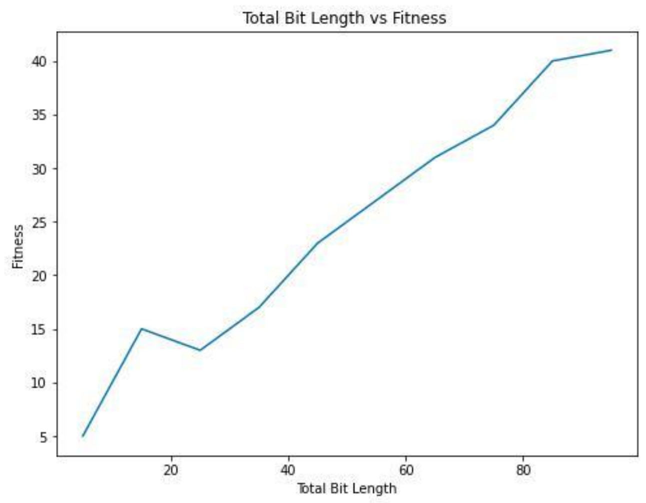

Fig. 20: Mimic: FourPeaks Problem Size vs. Fitness

For One Max problems, as the problem size grows, the search space also grows exponentially, leading to difficulty in generating high-quality solutions. Nevertheless, the MIMIC algorithm can still be effective, especially when combined with other techniques like restarts or hybridization with other algorithms.

In the case of 4-peaks problems, which involve a function with a plateau followed by a sharp peak, increasing the prob-lem size can improve the MIMIC algorithm's performance. The algorithm can efficiently explore plateaus and generate solutions near the peak using the probabilistic model.

Overall, the performance of the MIMIC algorithm for differ-ent optimization problems is determined by several factors, in-cluding the problem size, structure, and search strategy. While the MIMIC algorithm is a powerful tool for solving large-scale optimization problems, finding the optimal parameters and techniques may require experimentation and tuning.

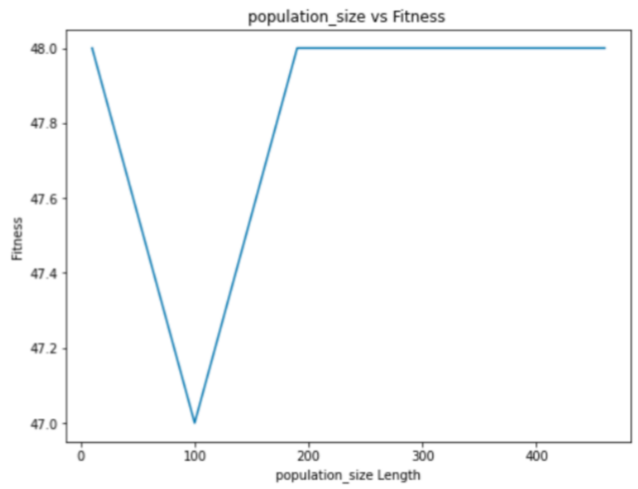

Fig. 21: Mimic: OneMax Problem Size vs. Fitness

V. Neural Network with Continuous values VI.

Randomized Hill Climbing (RHC)

My analysis focuses on the impact of the learning rate in Randomized Hill Climbing (RHC), which is an optimization algorithm commonly used for minimizing the loss function of a neural network (NN) by adjusting its weights and biases. The learning rate is a crucial parameter that determines the step size of the gradient descent algorithm, which updates the NN's parameters.

A high learning rate can result in overshooting the minimum of the loss function and causing the NN to diverge, leading to unstable solutions that do not converge to the global minimum. In contrast, a low learning rate can lead to slow convergence and suboptimal solutions. Hence, it is important to find the optimal learning rate that strikes a balance between stability and convergence speed.

To find the optimal learning rate, a common approach is to start with a small value and gradually increase it until the convergence speed slows down. This is known as a learning rate schedule, which allows RHC to explore the search space effectively initially and then converge more quickly to the minimum as it gets closer.

A higher learning rate can lead to faster convergence to a minimum, potentially improving the accuracy of the NN. However, it can also lead to instability and slower convergence, ultimately hurting the NN's accuracy. Therefore, finding the optimal learning rate that balances convergence speed and stability is crucial to maximizing the NN's accuracy.

In conclusion, the learning rate is a critical parameter that impacts the performance of RHC in terms of NN loss and accuracy. To find the optimal learning rate, a learning rate schedule can be used to strike a balance between convergence speed and stability.

VII. Simulated annealing

In my results, I observed that simulated annealing is an optimization algorithm that is commonly used to minimize the cost function of a neural network, where the cost function is the neural network's loss function. The algorithm's step size is controlled by the learning rate, which determines how quickly the algorithm explores the solution space. Choosing an appropriate learning rate is critical for successful neural network training. If the learning rate is too large, the algorithm may overshoot the global minimum and fail to converge. If it is too small, the algorithm may get stuck in a local minimum and fail to find the global minimum.

Several factors can affect the optimal learning rate for simulated annealing, including the problem being solved, the neural network's architecture, and the initial conditions. To determine the best learning rate, it is essential to perform a hyperparameter optimization technique, such as a grid search, to train the neural network with simulated annealing using a range of learning rates and choose the one that results in the best performance in terms of loss and accuracy.

In conclusion, the learning rate is a critical hyperparameter in simulated annealing for neural network training, and it can significantly impact the algorithm's performance. Balancing exploration and exploitation is necessary to achieve optimal performance in minimizing the cost function and finding the global minimum.

A. Genetic Algorithm

My study showed that genetic algorithms (GAs) are useful in training neural networks, and their performance can be affected by the population size, mutation rate, and learning rate. A larger population size can provide greater exploration of the solution space but increases the computational cost. Meanwhile, a higher mutation rate can promote thorough exploration of the solution space, but too much randomness in the process may lead to suboptimal results.

The learning rate is another crucial parameter that affects the performance of the optimization algorithm used to update the weights of the neural network. Setting the learning rate too high can cause overshooting of the minimum of the loss function, while setting it too low can lead to slow convergence.

To determine the optimal values for these parameters, a balance between exploration and exploitation must be achieved. A larger population size and higher mutation rate can facilitate thorough exploration of the solution space, while a smaller population size and lower mutation rate can focus on exploit-ing good solutions. The learning rate must also be fine-tuned to obtain the best neural network performance.

Ultimately, the optimal parameter values depend on the specific problem and neural network architecture. Experi-mentation and tuning are often required to obtain the best performance.

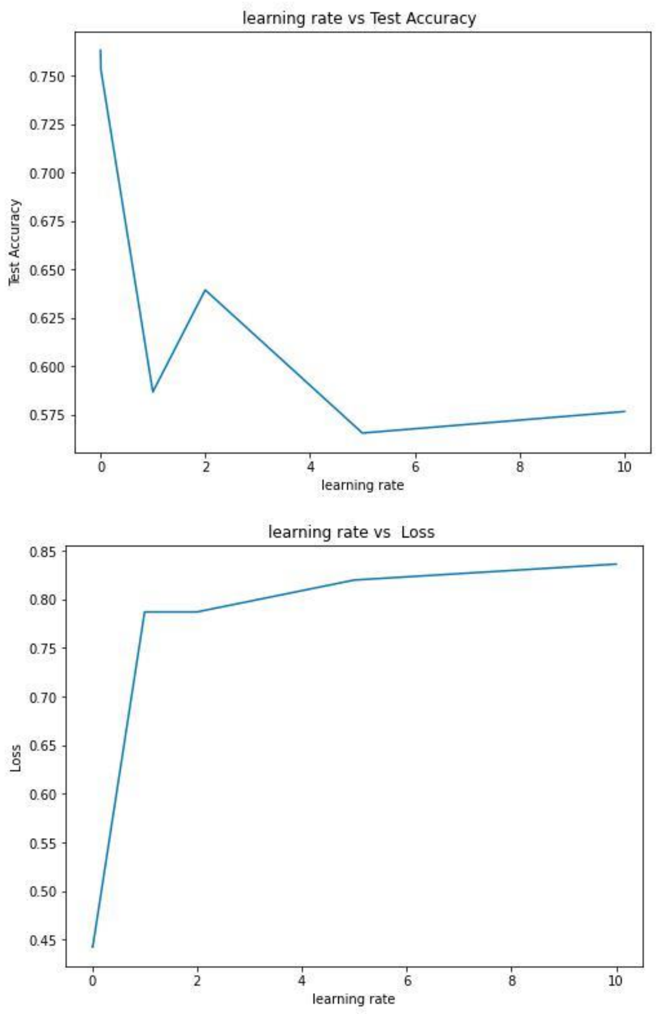

Fig. 22: Randomized hill climbing: Learning rate vs Test accuracy and loss

VIII. Comparison of local random search algorithms and Clock Time

Randomized hill climbing, simulated annealing, a genetic algorithm, and MIMIC are all local random search algorithms commonly used to solve optimization problems. While they share similarities in their basic approach of iteratively search-ing for better solutions, they differ in their implementation details, strengths, and weaknesses.

A. Randomized Hill Climbing:

1) Strengths: Randomized Hill Climbing is a simple and easy-to-implement algorithm that is memory-efficient and can quickly converge to a local optimum in small to medium-sized search spaces. However, it often gets stuck in local optima, has limited global search capabilities, and is sensitive to initialization, making it difficult to find a good starting point in high-dimensional search spaces.

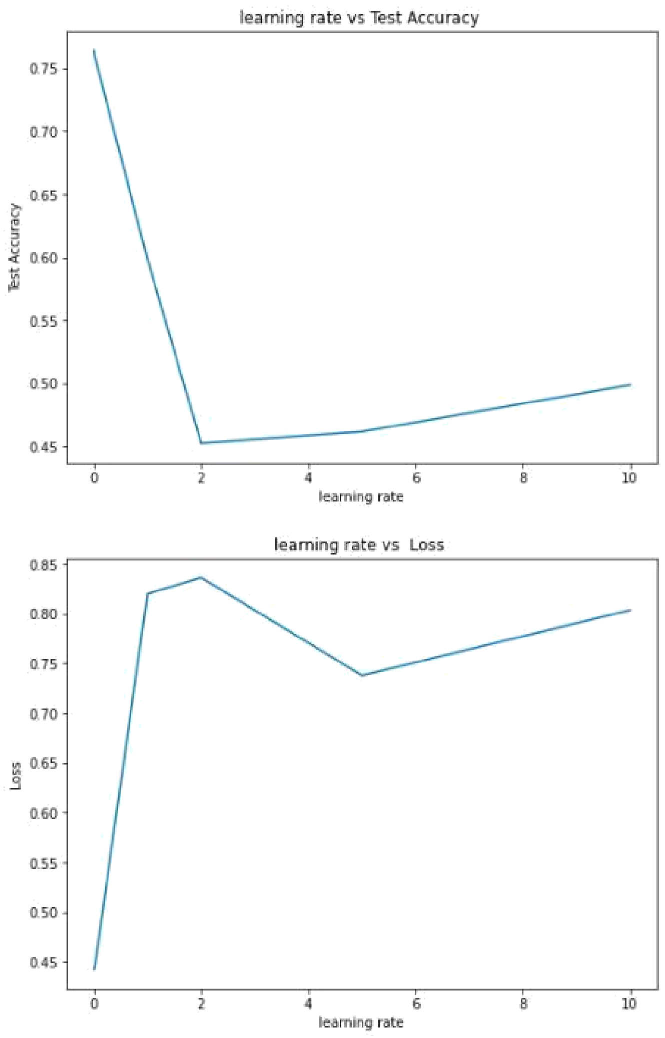

Fig. 23: Simulated annealing: Learning rate vs Test accuracy and loss

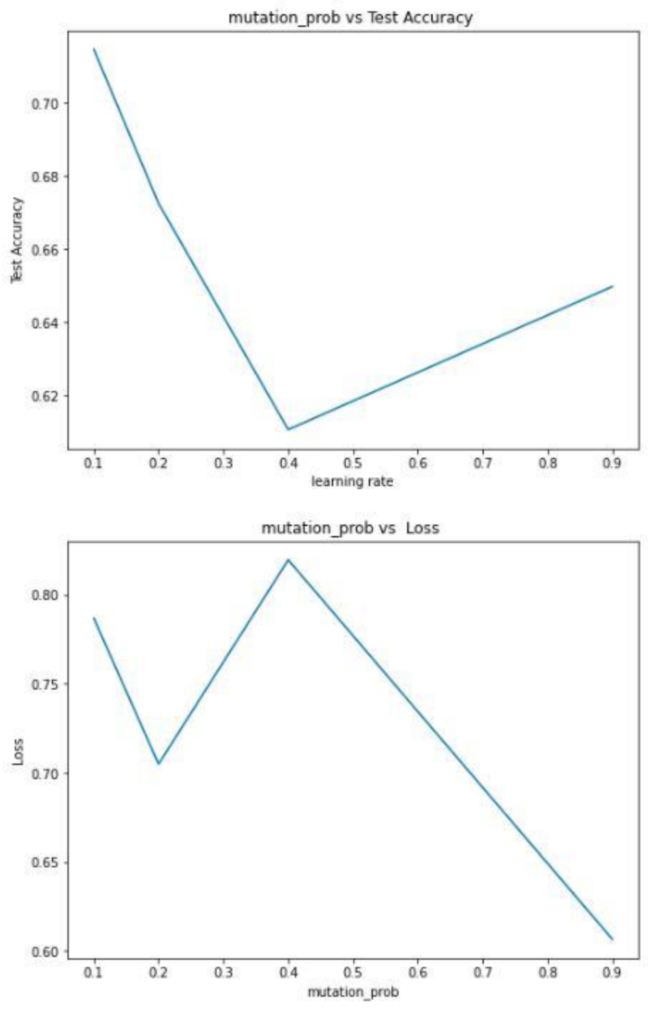

Fig. 24: Genetic algorithm: Mutation Rate vs Test accuracy and loss

B. Simulated Annealing

1) Strengths: Simulated annealing is a stochastic optimization algorithm that can explore the entire search space and find global optima by accepting worse solutions with a non-zero probability. It has a guaranteed convergence property under certain conditions and is robust to changes in problem formulation and noisy objective functions. However, simulated annealing is computationally expensive and requires a large number of function evaluations to converge to a good solution. The choice of temperature schedule affects its performance, and finding a good schedule can be challenging. Additionally, tuning parameters, such as the cooling rate and initial temperature, can also impact the algorithm's performance.

C. Genetic Algorithm

1) Strengths: The genetic algorithm is a population-based optimization algorithm that maintains a population of solutions and can explore multiple regions of the search space in parallel. It is robust to noise and can handle nonlinear and multimodal objective functions, and promotes diversity in the population, allowing it to escape local optima and explore the search space. However, it can be computationally expensive, requires a large number of function evaluations to converge to a good solution, and can converge prematurely to a suboptimal solution, especially in high-dimensional search spaces. The choice of parameters, such as population size and crossover probability, can also affect its performance.

D. MIMIC

MIMIC is a population-based algorithm that uses a probabilistic model and has a guaranteed convergence property to a global optimum. However, it is computationally expensive and memory-intensive, and is only effective for problems where dependencies among variables are present and can be modeled. The choice of algorithm depends on factors such as problem formulation, search space size, objective function, and computational resources available.

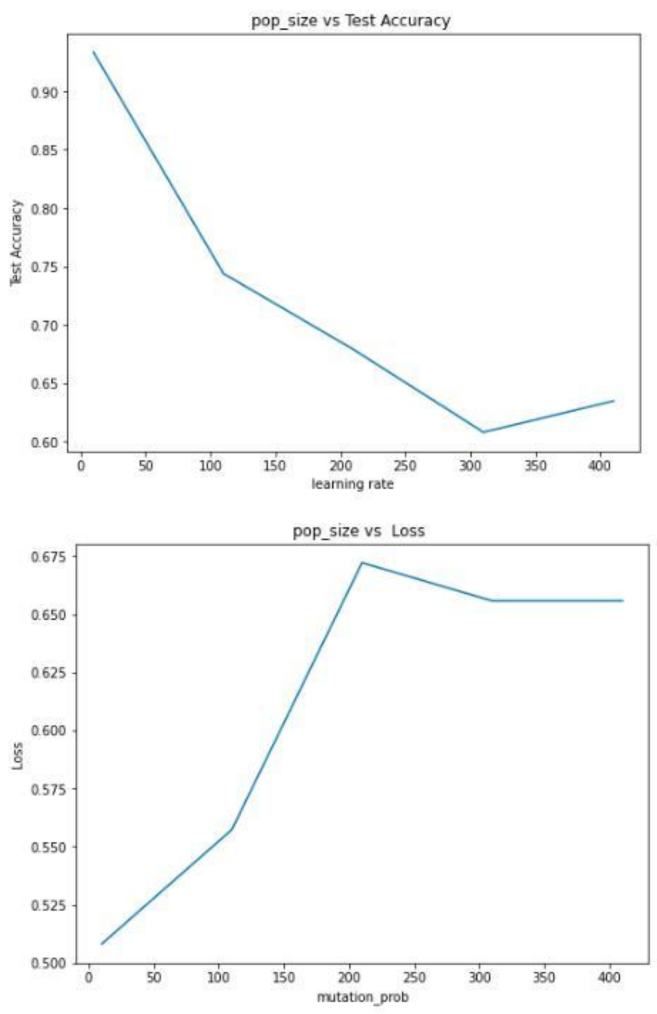

Fig. 25: Genetic algorithm: Population Size vs Test accuracy and loss

IX. Conclusion

This report summarizes the evaluation of several randomized optimization algorithms including randomized hill climbing, simulated annealing, a genetic algorithm, and MIMIC on different optimization problems. The study analyzed how various parameters, such as population size, mutation rate, learning rate, and problem size, affect the performance of these algorithms in terms of optimization quality, running time, and robustness. The study showed that each algorithm has its own strengths and limitations and that their optimal parameter values depend on the problem instance and the characteristics of the optimization landscape. Therefore, hyperparameter tuning and algorithm selection are crucial in designing effective optimization algorithms. The report also emphasizes the importance of balancing exploration and exploitation, avoiding premature convergence, and adapting to the problem at hand. Overall, randomized optimization algorithms are powerful and versatile tools for solving a wide range of optimization problems in various domains, and understanding their strengths and limitations can help in developing novel algorithms that combine their strengths.