Machine Learning
Randomized Optimization

Jorge Salvador Aguilar Moreno

***Abstract*—This report presents the implementation and analysis of three local random search algorithms—Randomized Hill Climbing (RHC), Simulated Annealing (SA), and a Genetic Algorithm (GA)—on two discrete-valued optimization problem domains: 1) The 4-peaks problem highlighting the advantages of GA, and 2) the max k-color problem highlighting the advantages of SA. Additionally, these algorithms are applied to optimize the weights of a neural network, replacing traditional backpropagation.**

***Index Terms*—supervised learning, machine learning, random search, hill climbing, simulated annealing, genetic algorithm, neural network.**

I. Introduction

Random search algorithms are optimization techniques that explore the solution space by generating random candidate solutions. These methods are useful for complex, high-dimensional problems where traditional approaches may struggle, such as with non-differentiable or multi-modal functions. Random search helps avoid local optima by introducing randomness, making them effective for global optimization [1]. Common random search algorithms include randomized hill climbing, simulated annealing, and genetic algorithms.

- **Randomized Hill Climbing (RHC)**: local search optimization algorithm that starts with a randomly chosen solution and iteratively makes small, random changes to it. If the change improves the objective function, it is accepted as the new solution; otherwise, it is rejected. RHC can efficiently explore the search space in simple problems, but its tendency to get stuck in local optima is a major limitation, especially in complex or multimodal landscapes [2].
- **Simulated Annealing (SA)**: Optimization algorithm inspired by the annealing process in metallurgy, where it probabilistically accepts worse solutions to escape local optima. As the algorithm progresses, this acceptance probability decreases, allowing for exploration early on and exploitation later. This trade-off between exploration and exploitation makes SA highly suitable for complex optimization problems with multiple local minima [3].
- **Genetic Algorithm (GA)**: Algorithms based on the concept of natural selection and evolution. GA operates on a population of solutions and evolves the population using genetic operators such as mutation, crossover, and selection. The diversity in the population helps prevent premature convergence to suboptimal solutions, making GAs well-suited for large, multimodal search spaces [4].

The aim of this report is to explore the effectiveness of randomized optimization techniques by applying several random search algorithms. The report is divided into two main parts:

- **Part 1:** The implementation of RHC, SA, and GA on two different optimization problems: the 4-Peaks problem and the Max K-Color problem. Both problems have a defined fitness function that aims to be maximized, allowing for a comparison of how different algorithms perform. Bit strings are used to simplify the optimization process.
- **Part 2:** The application of RHC, SA, and GA to find optimal weights for a neural network. These randomized optimization techniques are compared to the traditional backpropagation approach to assess their effectiveness in neural network training.

These three algorithms—RHC, SA, and GA—are particularly interesting for randomized optimization (RO) because they each introduce unique mechanisms for navigating complex search spaces. RHC is a simple yet efficient local search method that can perform well when applied to problems with relatively smooth landscapes, but it struggles with local optima. SA improves upon this by introducing a probabilistic mechanism to escape local optima, making it more effective in highly complex, multimodal landscapes. GA, with its population-based approach and genetic operators, is well-suited for large, multimodal search spaces where maintaining diversity among solutions is essential for avoiding premature convergence. The combination of these different strategies makes them valuable in RO, offering varied approaches to balance exploration and exploitation in search of global optima [1].

II. Part 1A: 4-Peaks Problem

The 4-Peaks problem challenges algorithms to balance between exploring local optima and exploiting global optima. Defined on a bit string of length n, the goal is to maximize the number of leading 1s and trailing 0s. The fitness function is:

$$\text{Fitness}(x, T) = \max(\text{tail}(0, x), \text{head}(1, x)) + R(x, T)$$

Where:

- $\text{tail}(0, x)$ counts the trailing 0s.
- $\text{head}(1, x)$ counts the leading 1s.
- $R(x, T)$ rewards the solution with n if both counts exceed a threshold T, otherwise 0.

The threshold T is a fraction of n, given by $T = t_{\text{pct}} \times n$, where t_{pct} is typically 0.1. The maximum fitness is achieved when both head$(1, x)$ and tail$(0, x)$ exceed T:

$$\text{Maximum fitness} = \max(n - T, T) + n$$

For example, the state $[1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0]$ has head$(1, x) = 3$ and tail$(0, x) = 5$. Since both exceed $T = 3$, the total fitness is:

$$\text{Fitness}(x, T) = 5 + 12 = 16.$$

A. Hypothesis

In the 4-Peaks problem, we hypothesize that GA will outperform RHC and SA in terms of finding the global optima more consistently. This is due to GA's ability to explore a large portion of the solution space by maintaining a diverse population of solutions. Through the use of genetic operators like crossover and mutation, GA can combine parts of different solutions to discover better ones, making it more likely to avoid local optima and efficiently explore multimodal landscapes. SA, with its controlled randomization and ability to accept worse solutions, is expected to perform better than RHC but may not be as efficient as GA for larger search spaces.

In terms of speed, we expect RHC to be the fastest algorithm since it follows a simple local search process without maintaining a population or performing complex operations like crossover. However, RHC's tendency to get stuck in local optima will likely make it less effective in solving the problem compared to SA and GA. Simulated Annealing, due to its gradual cooling schedule, may take longer than RHC but can explore a larger area of the solution space, potentially leading to better solutions at the cost of increased runtime. GA is expected to take the longest time due to the population-based search mechanism and the additional computational cost of genetic operators.

B. Experimental Methodology

The RO implementation uses the Python library `mlrose-hiive`. The goal was to keep the three algorithms with a same number of maximum attempts and maximum iterations so that they could be compared fairly. To account for variability in the input, ten different seeds were applied to the problem. This approach ensures a robust evaluation of the performance of each algorithm by considering multiple initial conditions and capturing potential variations in their results.

The algorithms were implemented as follows:

- **Randomized Hill Climbing:**

```
mlrose.random_hill_climb(problem, random_state=
    seed, max_attempts=10, max_iters=200,
    restarts=10, curve=True)
```

- **Simulated Annealing:**

```
mlrose.simulated_annealing(problem, random_state
    =seed, max_attempts=10, max_iters=200,
    schedule=mlrose.ExpDecay(init_temp=0.1,
    exp_const=0.0005,min_temp=0.001),curve=True)
```

- **Genetic Algorithm:**

```
mlrose.genetic_alg(problem, random_state=seed,
    pop_size=100, mutation_prob=0.1,
    max_attempts=10, max_iters=200, curve=True)
```

The hyperparameters of the optimization algorithms were manually tuned to achieve satisfactory performance. For example, in Simulated Annealing (SA), an exponential decay schedule was selected as it consistently outperformed other schedules. For the Genetic Algorithm (GA), a population size of 100 with a mutation probability of 0.1 provided balanced results, yielding competitive scores without requiring a significant increase in population size, which would only lead to marginal improvements. These tuning decisions were aimed at optimizing performance while maintaining computational efficiency.

C. Results from Algorithms

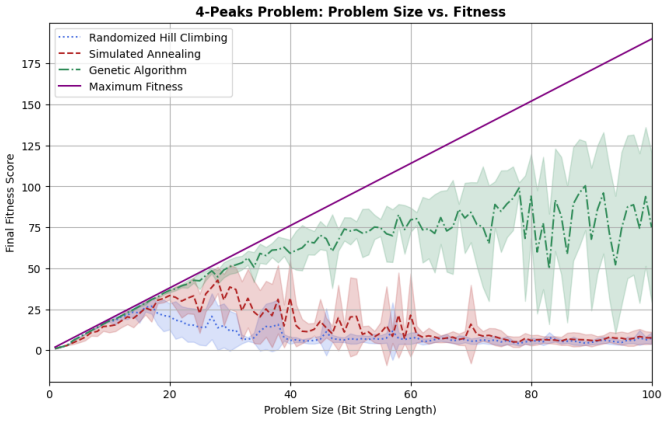

Fig. 1. 4-Peaks problem: Fitness scores for different problem sizes

The Figure 1 highlights the performance advantage of GA for the 4-Peaks problem for all problem sizes. The final fitness scores of the different RO algorithms are compared as the problem size increases. The problem sizes correspond to the length of the bit strings used as inputs. With `max_attempts = 10` and `max_iters = 200`, all algorithms perform well for small problems of up to 15 bits. However, as the problem size increases, GA significantly outperforms the other algorithms. Around a problem size of 60 bits, GA's performance stabilizes, oscillating around a fitness score of 75. Nevertheless, tuning key hyperparameters, such as `pop_size` and `max_attempts`, would improve GA's performance for larger bit strings, leading to higher fitness scores.

SA demonstrates superior performance compared to RHC for medium-sized problems, particularly those in the 20 to

40-bit range. However, as the problem size increases, the fitness scores of SA and RHC become more comparable. While tuning hyperparameters such as `max_attempts` and `schedule` could further improve SA's performance, it is unlikely that either algorithm will achieve fitness scores comparable to those of GA.

The baseline problem size for the rest of RO algorithm comparisons using the 4-Peaks problem is 30 bits as input, which has a maximum fitness score of 57.

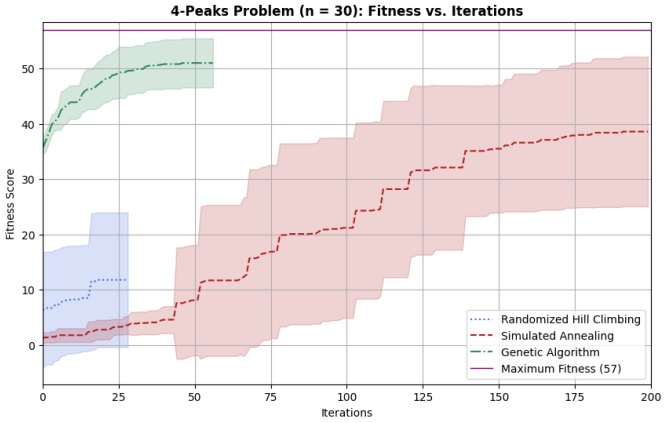

Fig. 2. 4-Peaks problem: Variation of fitness score for each iteration

Figure 2 illustrates how the fitness score evolves with the number of iterations. GA rapidly outperforms SA and RHC in the initial iterations, showcasing one of its key advantages: the ability to explore multiple solutions simultaneously through genetic diversity. After converging, GA achieves a score of 52, approaching the maximum possible fitness of 57. Another notable aspect is that GA demonstrates less variability in fitness scores across iterations compared to the other algorithms. This is because GA operates on a population of solutions, and through selection, crossover, and mutation, it mitigates the risk of getting stuck in suboptimal regions of the search space, leading to more consistent results.

SA exhibits the most significant fitness improvement over the course of iterations and requires the most iterations to converge. Nevertheless, given enough iterations, it emerges as the second-best performing algorithm. Finally, RHC initially surpasses SA but quickly plateaus and underperforms compared to both SA and GA, as it is prone to getting trapped in local optima and lacks mechanisms to effectively escape them.

Figure 3 and Figure 4 depict the relationship between the number of iterations and the cumulative function evaluations for the different algorithms. The plot demonstrates how computationally intensive each algorithm is by showing the total number of function evaluations over time. As expected, GA shows the highest number of function evaluations due to its population-based approach and a large variability when it is approaching convergence. At each iteration, GA evaluates multiple solutions (individuals) in parallel, applying crossover and mutation operators, which results in a steep increase in cumulative evaluations. This makes GA computationally

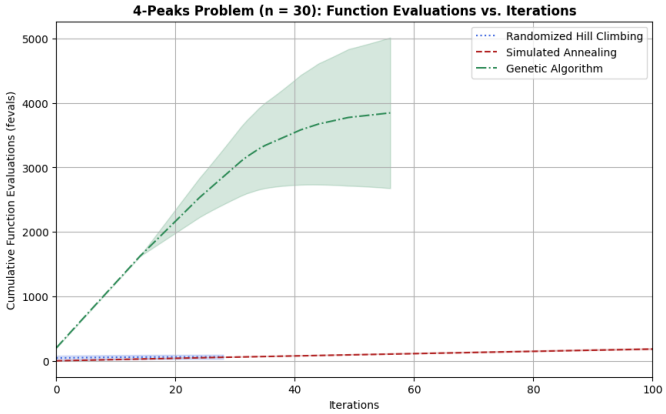

Fig. 3. 4-Peaks problem: Number of fevals for each iteration

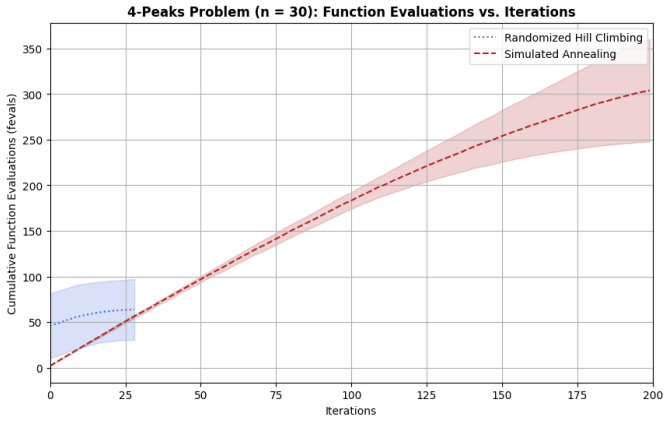

Fig. 4. 4-Peaks problem: Number of fevals for each iteration (excluding GA)

expensive but also highly effective in exploring large search spaces and escaping local optima. At the end, the GA curve shows a characteristic plateau because the population tends to converge towards similar or identical solutions. As the diversity within the population decreases, the likelihood of generating significantly new or improved solutions diminishes. The average final cumulative function evaluations for GA is 3932.

Figure 4 shows the same plot as Figure 3 but excluding GA. In this Figure, SA demonstrates a more moderate increase in cumulative evaluations. SA probabilistically accepts worse solutions early in the search process to avoid local optima and explores a broader area of the solution space. As the number of iterations increases, the acceptance of worse solutions gradually decreases per the exponential temperature decay, allowing the algorithm to converge. While SA performs more function evaluations than RHC, it balances exploration and exploitation efficiently. RHC performs the fewest evaluations, as it only evaluates a single neighboring solution at each iteration. RHC's simple and localized search approach results in a much slower accumulation of function evaluations. However, RHC is more prone to getting stuck in local optima, which limits its ability to find global solutions, especially in

more complex problem spaces. The average final cumulative function evaluations for SA is 304, and 62 for RCH.

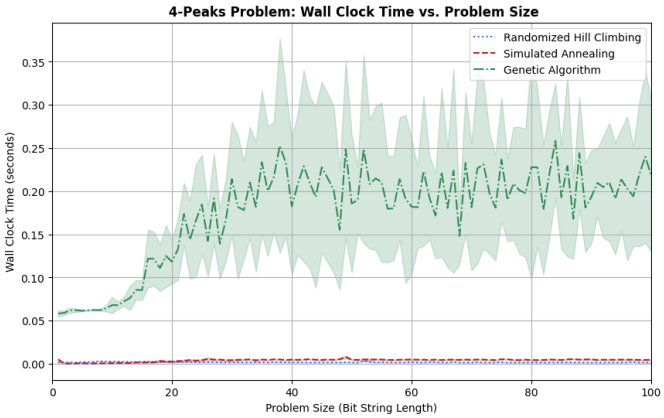

Fig. 5. 4-Peaks problem: Wall clock time for different problem sizes

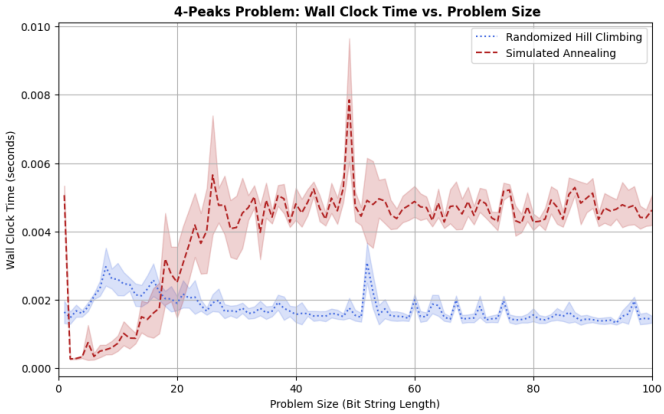

Fig. 6. 4-Peaks problem: Wall clock time for different problem sizes (excluding GA)

Figure 5 and Figure 6 illustrates the relationship between wall clock time and problem size for the various optimization algorithms. As expected, the wall clock time increases with the problem size for all algorithms. GA consistently requires the most time, particularly for larger problem sizes, due to the overhead of maintaining and evolving a population through crossover, mutation, and selection operations. This computational time for GA is one of the main drawbacks of this algorithm. GA takes 0.06 s to solve small problems up to 10 bits, and takes about 0.20 s to solve problem sizes over 30 bits using the current implementation. SA follows but with drastically less time.

Zooming in the SA and RHC results in Figure 6, SA involves probabilistic acceptance of worse solutions, leading to longer exploration phases, especially for complex problems. For problems over 20 bits, the algorithm typically takes 0.0045 s to convege. RHC, being a simpler local search method, completes in the shortest time. However, its faster run time comes at the cost of lower solution quality for larger problem sizes, as seen in other performance metrics. This algorithm takes approximately 0.002 s to converge using the current implementation.

III. Part 1B: Max k-Color Problem

In the Max K-Color problem, the goal is to assign colors to the vertices of a graph such that no two adjacent vertices share the same color, while using at most K colors. When $K = 2$, the problem becomes a binary coloring problem, meaning each vertex is assigned one of two possible colors, typically represented as 0 or 1.

As a maximization problem, the objective is to maximize the number of edges between vertices that are assigned different colors. This ensures that adjacent vertices are not colored the same. The problem can be encoded as a bit string, where each bit in the string corresponds to a vertex in the graph, and the value of the bit (either 0 or 1) represents the color assigned to that vertex.

The fitness function for the Max K-Color problem with two colors is defined as the number of edges in the graph that connect vertices of different colors. Formally, for a given bit string x, which represents the color assignment, the fitness function $f(x)$ is:

$$f(x) = \sum_{(i,j)\in E} \mathbb{1}(x_i \neq x_j)$$

where E is the set of edges in the graph, and $\mathbb{1}(x_i \neq x_j)$ is an indicator function that returns 1 if the colors assigned to vertices i and j are different, and 0 otherwise. The goal is to maximize $f(x)$, which corresponds to maximizing the number of properly colored edges in the graph.

A. Hypothesis

In the Max K-Color problem, we hypothesize that SA will outperform GA and RHC in terms of consistently finding better solutions. SA's ability to probabilistically accept worse solutions early in the search enables it to escape local optima, which is crucial for problems like Max K-Color, where the solution space contains many local traps. The cooling schedule in SA allows for exploration in the initial stages of the algorithm, gradually shifting towards exploitation as the temperature decreases, which makes it well-suited to navigating complex, multimodal solution spaces. GA, although effective at maintaining diversity in the population through crossover and mutation, may struggle to outperform SA due to the additional overhead required to balance exploration and exploitation. GA's performance heavily depends on careful tuning of its population size and mutation rates.

In terms of speed, we expect RHC to be the fastest algorithm since it performs a simple greedy search that accepts improvements without requiring additional operations like crossover or mutation. However, RHC is likely to get stuck in local optima, especially for larger graphs with many conflicting edges, making it less effective in solving the Max K-Color problem. Simulated Annealing, due to its controlled randomization and the gradual cooling schedule, is expected to take longer than RHC but should deliver better-quality solutions. On the other

hand, GA is anticipated to be the slowest algorithm due to its population-based search and computationally expensive genetic operations like crossover and mutation. While GA may explore a diverse range of solutions, the time complexity will likely be higher compared to both SA and RHC.

B. Experimental Methodology

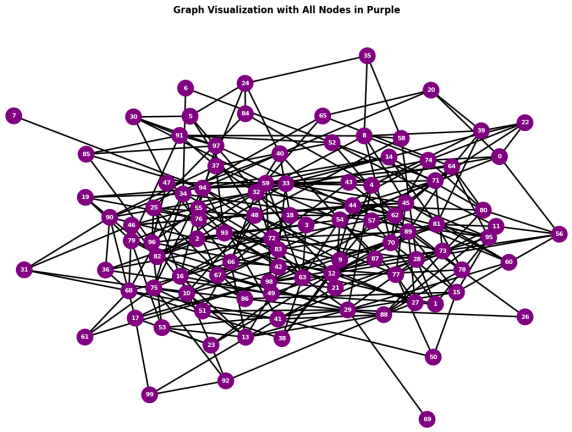

Fig. 7. Max k-Color problem: Graph to be solved (100 nodes with 5 or less edges)

The implementation involves solving the Max K-Color problem, as illustrated in the graph shown in Figure 7. The graph consists of 100 nodes, with each node having up to 5 connections. The objective is to maximize the number of edges connecting nodes of different colors, using only two colors. This ensures that adjacent nodes have distinct colors, thereby maximizing the number of valid, non-conflicting edges in the graph.

The RO implementation for the Max K-Color problem follows a similar approach to that used for the 4-Peaks problem. The Python library `mlrose-hiive` was utilized, and consistent maximum attempts and maximum iterations were maintained across the three algorithms to ensure a fair comparison. However, since the Max K-Color problem is inherently more complex and requires a broader exploration of the search space, the maximum limits for iterations and attempts were increased accordingly. To account for variability in the input, five different seeds were applied to the problem. The algorithms were implemented as follows:

- **Randomized Hill Climbing:**

```
mlrose.random_hill_climb(problem, random_state=
    seed, max_attempts=100, max_iters=3000,
    restarts=10, curve=True)
```

- **Simulated Annealing:**

```
mlrose.simulated_annealing(problem, random_state
    =seed, max_attempts=100, max_iters=3000,
    schedule=mlrose.ExpDecay(init_temp=200,
    exp_const=0.01,min_temp=0.00001),curve=True)
```

- **Genetic Algorithm:**

```
mlrose.genetic_alg(problem, random_state=seed,
    pop_size=100, mutation_prob=0.1,
    max_attempts=100, max_iters=3000,curve=True)
```

C. Results from Algorithms

This section presents similar plots to those from the 4-Peaks problem, with the aim of emphasizing the advantages of SA over the other algorithms in terms of both performance and time in the Max K-color problem. The comparison demonstrates how SA's probabilistic exploration allows it to navigate complex solution landscapes more effectively than RHC, while achieving faster convergence compared to the more computationally expensive GA.

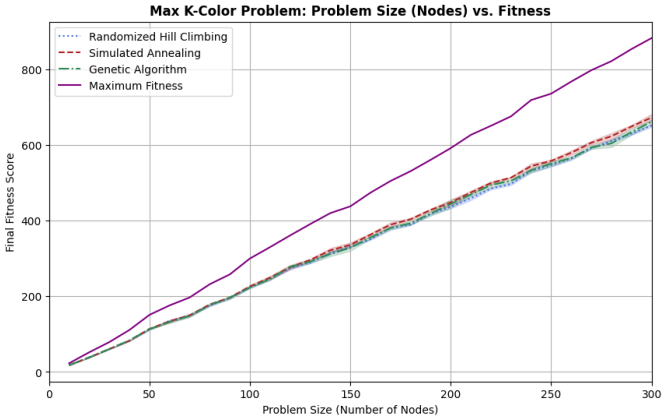

Fig. 8. Max k-Color problem: Fitness scores for different problem sizes

Figure 8 illustrates the fitness scores of each algorithm as the problem size increases and compares it to the maximum fitness, defined as the number of edges (in some cases, it is impossible to achieve this score, as it depends on the graph layout). While all algorithms perform similarly for smaller problem sizes, SA consistently demonstrates marginally better results compared to the others. As the problem size grows, the algorithms deviate further from the maximum achievable fitness score. This divergence can be mitigated by tuning hyperparameters. For example, the performance of SA could be improved by adjusting the cooling schedule to decrease the temperature more gradually, or by increasing the number of iterations, allowing for better exploration of the solution space before convergence.

As a basis for comparison, the graph shown in Figure 7 is used to implement the RO algorithms and report results. The objective is to highlight the benefits of SA over the other algorithms for this specific problem.

Figure 9 presents the fitness score as a function of the number of iterations. As anticipated, SA shows a low fitness core at initial stages, and requires the highest number of iterations to converge, whereas GA converges after the least iterations. Notably, SA achieves a marginally higher fitness score compared to GA and RHC. Upon convergence, the

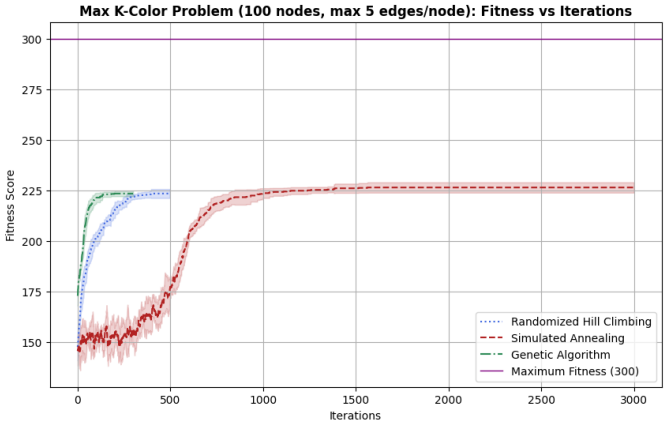

Fig. 9. Max k-Color problem: Variation of fitness score for each iteration

variability in fitness scores across all three algorithms is similar. However, the average fitness score for SA is 226.4, while GA and RHC remain just below the threshold of 225.

For visualization purposes, Figure 10 displays the best solution of the Max K-color problem corresponding to the implementation of SA. In this graph, green edges represent pairs of nodes that meet the fitness criteria, achieving a score of one. The graph shows that 230 edges satisfy the fitness function.

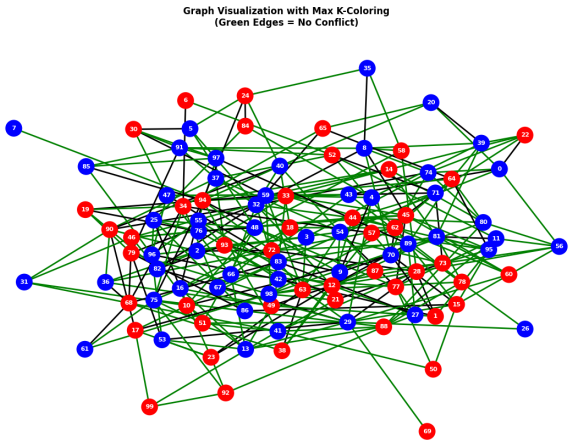

Fig. 10. Max k-Color problem: Best solution with SA algorithm

Figure 11 illustrates that SA requires the fewest function evaluations (fevals) per iteration; however, it also necessitates more iterations to reach convergence. The characteristic shape of SA's curve is influenced by the exponential decay in temperature, a core feature of the SA implementation. On the other hand, RHC converges rapidly and requires relatively few fevals, demonstrating its ability to quickly reach a solution. In contrast, GA exhibits a steep curve, demanding a significant number of fevals at each iteration. Additionally, GA shows a large variance as it approaches convergence, reflecting the diversity within its population-based search mechanism.

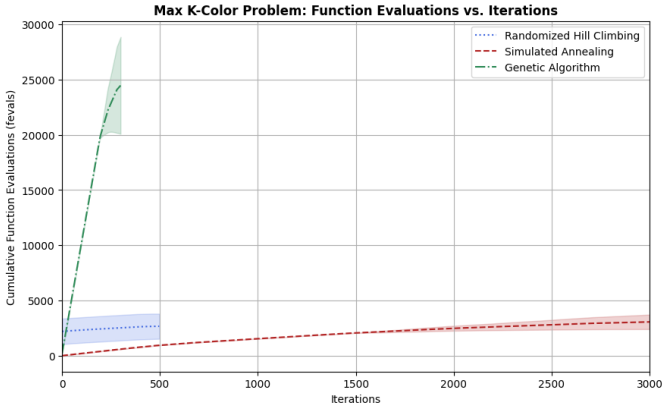

Fig. 11. Max k-Color problem: Number of fevals for each iteration

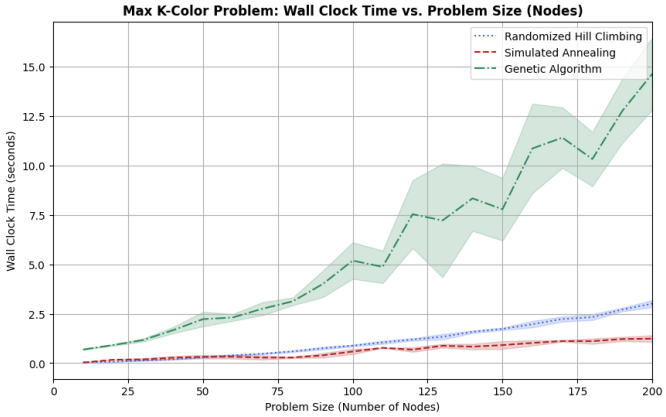

Fig. 12. Max k-Color problem: Wall clock time for different problem sizes

Figure 12 highlights one of the key advantages of SA for the Max K-color problem — its computational efficiency. Despite all the algorithms converging to similar solutions, with SA providing marginally better results, it is also the fastest to reach this solution. This efficiency is due to the controlled cooling schedule of SA, which allows for a gradual exploration of the solution space while reducing unnecessary evaluations. RHC is the second-best in terms of computational time, though it requires roughly double the time taken by SA to converge. Finally, GA is by far the slowest algorithm, mainly due to the population-based approach, which involves more complex operations like selection, crossover, and mutation at each iteration. Given the similar results across all algorithms, GA becomes the least preferred option for this Max K-color problem due to its high computational cost and similar results.

IV. Part 2: Random Optimization for Neural Network Weights

This section applies three local random search algorithms—RHC, SA, and GA—to optimize the weights of a neural network. Traditionally, neural networks rely on backpropagation with gradient descent to update their weights and minimize the loss function. Backpropagation computes the gradient

of the loss with respect to each weight and adjusts them iteratively using gradient descent. While effective, backpropagation can sometimes struggle with local minima or saddle points, especially in complex optimization landscapes. The use of RO algorithms offers an alternative approach to weight optimization by exploring the weight space without relying on gradient information.

In the context of neural networks, weight optimization poses a unique challenge because the weights are continuous and real-valued, as opposed to the discrete solution spaces often encountered in combinatorial optimization problems. RO algorithms are typically well-suited for problems where the search space is discrete. However, when applied to neural networks, these algorithms must explore a continuous weight space, making their task more complex. Instead of flipping bits or swapping discrete values, RO algorithms must now fine-tune real-valued parameters, which can involve significantly more nuanced exploration of the solution space. The continuous nature of neural network weights requires careful handling of step sizes, temperature schedules, and mutation rates to ensure effective search dynamics. This complexity can affect both the convergence speed and the quality of solutions that these algorithms provide when applied to neural networks.

A. Diabetes Dataset

The following dataset is used as the basis for implementing the neural network:

1) **Diabetes dataset** (Kaggle). This dataset contains 768 instances and 8 features: pregnancies, glucose level, blood pressure, skin thickness, insulin, BMI, diabetes pedigree function, and age.

The diabetes dataset is a binary classification problem, aiming to predict diabetes presence or absence. This dataset is not on the prohibited list in the assignment. Although `sklearn.datasets` includes a diabetes dataset, the one used in this report contains entirely different information.

B. Hypothesis

We hypothesize that the standard backpropagation with gradient descent will outperform the local random search algorithms (RHC, SA, and GA) in terms of both convergence speed and final accuracy. Backpropagation, which directly computes the gradient of the loss function, is highly efficient in minimizing error and is well-suited for the smooth, continuous weight space of neural networks. By iteratively updating weights in the direction of steepest descent, backpropagation is expected to converge quickly to an optimal or near-optimal solution.

We expect the randomized optimization algorithms to show varying levels of performance. GA is expected to explore the weight space more thoroughly, potentially yielding superior solutions. By maintaining a population of solutions and using genetic operations like crossover and mutation, GA can explore multiple regions simultaneously, reducing the chances of getting trapped in local minima. However, this comes at a cost: GA is likely to have the longest runtime. SA is anticipated to outperform RHC. SA's ability to escape local minima by probabilistically accepting worse solutions gives it an edge over RHC, which is prone to premature convergence. However, SA may require more iterations and longer runtime due to its gradual exploration and slower convergence as the temperature decreases. RHC, while fast and computationally inexpensive, is likely to be the least effective due to its tendency to get stuck in suboptimal solutions.

C. Experimental Methodology

The neural network implementation for this study uses the Python library `pyperch`. The hyperparameters for the standard backpropagation implementation are consistent with those used in Assignment 1, which were manually tuned to achieve the highest score. The RO algorithms were implemented with comparable maximum attempts and iterations to ensure fair comparison.

The network consists of two hidden layers with 8 nodes each, and the activation function for Gradient Descent and RHC is sigmoid, while GA uses SELU. Gradient Descent utilizes the Adam optimizer, while RHC and SA employ their respective local search algorithms. SA's performance is influenced by an annealing schedule with gradual cooling. GA evolves a population of solutions with genetic operators such as mutation and crossover, with a population size of 300, mating pool of 30, and mutation rate of 10 individuals per generation. All algorithms were set to run for up to 5000 epochs (fewer for GA), and early stopping was implemented to avoid overfitting.

D. Results from Algorithms

TABLE I
SUMMARY OF PERFORMANCE FOR DIFFERENT ALGORITHMS TO DETERMINE NN WEIGHTS

Algorithm	Backprop	RHC	SA	GA
Training Accuracy	0.7818	0.7997	0.7394	0.7801
Test Accuracy	0.7792	0.7143	0.8052	0.7338
Execution Time (s)	73.9617	44.0729	39.8119	1104.5106

Table 1 summarizes the performance of four algorithms in finding weights a neural network. SA achieved the highest test accuracy at 80.52%, outperforming both Backpropagation (77.92%) and GA (73.38%), indicating its effectiveness in finding better solutions in this context. Interestingly, RHC showed the highest training accuracy at 79.97%, but its test accuracy was the lowest, suggesting overfitting. In terms of execution time, SA was the fastest, taking only 39.81 seconds, followed closely by RHC at 44.07 seconds, while GA was by far the slowest, requiring over 1100 seconds due to its more complex population-based search method. Backpropagation offered a good balance, with reasonable accuracy and a moderate execution time of 73.96 seconds.

Figure 13 presents the iterative learning curves, with iteration numbers on the x-axis and the cross-entropy loss (log-loss) on the y-axis. The left-hand plot illustrates the results for the training data, while the right-hand plot shows the

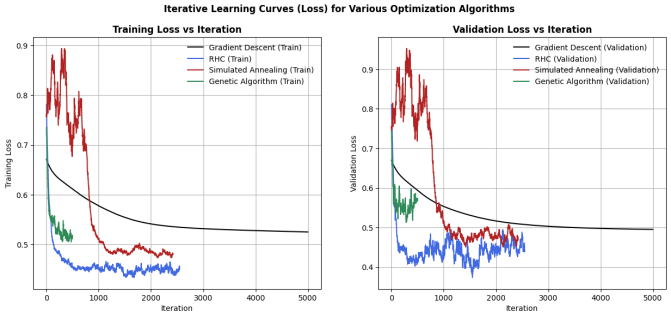

Fig. 13. Iterative learning curves for loss: training data (left), and validation data (right)

validation data. As anticipated, the gradient descent algorithm demonstrates a gradual reduction in log-loss for both the training and validation datasets. RHC quickly reduces the loss within the first few iterations; however, as highlighted in Table 1, it overfits the training data, and this low value of loss is not necessarily a good indicator. SA, on the other hand, takes approximately 800 iterations to make significant improvements, but once stabilized, the log-loss oscillates around 0.5 for both plots, which is a desirable value. Finally, the GA exhibits a steep reduction in log-loss during the initial iterations. After stabilization, the training loss settles at around 0.52, while the validation loss levels off at approximately 0.58. Despite the small number of iterations to converge, GA is the algorithm with longest execution time by far, as shown in Table 1.

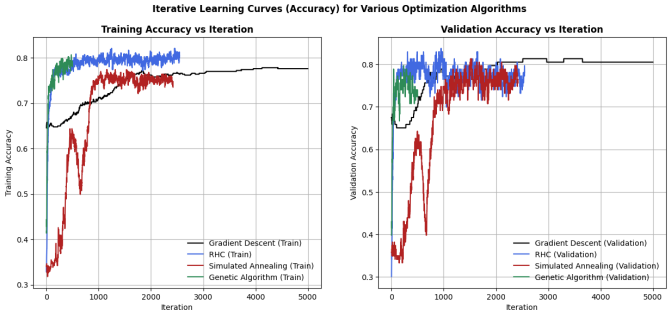

Fig. 14. Iterative learning curves for accuracy: training data (left), and validation data (right)

Figure 14 presents another iterative learning curve, this time with accuracy on the y-axis. As before, the left plot corresponds to the training data, while the right plot shows the validation data. Due to the simplicity of this binary classification problem, the validation accuracy occasionally exceeds the training accuracy. For gradient descent, the results mirror those from Assignment 1, with a gradual improvement in accuracy until flattening around iteration 2000. RHC quickly achieves high accuracy in the training data; however, in the validation data, the accuracy drops below 0.8, indicating overfitting. For SA, the algorithm takes approximately 1000 iterations to stabilize at an accuracy of around 0.75 in both the training and validation data, which is a positive indicator of performance. GA, similar to RHC, quickly reaches a high accuracy of approximately 0.78 in the training data, but in the validation data, the convergence score is lower, around 0.72, and this illustrates a heavier reliance on the training data. These results align with the performance summary in Table 1.

V. Conclusion

The results of this report demonstrate the varying strengths and weaknesses of randomized optimization algorithms when applied to different types of problems. In the 4-Peaks problem, the GA clearly outperformed RHC and SA, particularly for larger problem sizes. GA's population-based approach allowed it to consistently avoid local optima and find high-quality solutions, making it the most effective algorithm for this type of optimization problem. This result is consistent with the hypothesis for this problem.

In contrast, the Max K-Color problem was most effectively solved using SA. SA's ability to probabilistically escape local optima during the early stages of the algorithm allowed it to outperform both RHC and GA. The controlled exploration provided by the cooling schedule made it the most reliable approach for this particular problem. The superior performance of SA for the Max K-Color problem aligns with the hypothesis for this problem

While evaluating performance, it is essential to consider not just the accuracy or fitness score but also factors such as execution time and the overall computational cost. For example, although GA performed exceptionally well in the 4-Peaks problem, its computational expense was significantly higher than the other algorithms, especially for large problem sizes. Similarly, in the Max K-Color problem, SA achieved the best balance between performance and execution time.

Finally, when applying these randomized optimization algorithms to find neural network weights, the results show that they can produce outcomes comparable to, and in some cases better than, traditional backpropagation. SA, in particular, delivered test accuracies higher than backpropagation. This result was surprising based on the hypothesis for this section, as it was not expected RO would perform as well as the standard backpropagation implementation. This may be partly due to the simplicity of the binary classification task used in this report, but it highlights the potential of combining multiple machine learning techniques to achieve more optimal solutions. Randomized optimization techniques, when properly tuned and applied, can offer robust alternatives or supplements to more conventional methods in machine learning.

References

[1] Spall, J. C. (2003). *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*. John Wiley & Sons.
[2] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.
[3] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
[4] Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach*. Pearson.