Randomized Optimization

Kirsten Odendaal

I. Introduction

Conventional deterministic optimization methods often struggle with real-world problems as they conventionally contain non-differentiable functions, large input spaces, and noisy functions with many local optima. These challenges can cause traditional optimizers to become "stuck" in sub-optimal solutions or prove to be strictly infeasible [10]. To address these limitations, this paper explores random optimization methods.

In this study, three algorithms are explored: Random Hill Climbing (RHC), Simulated Annealing (SA), and Genetic Algorithms (GA). These methods are applied to three use cases, from academic to real-world. Initially, two unique optimization benchmark problems, *FourPeaks (FP)* and *Max k-Colour (KC)*, are examined to gain intuition on the behaviour and performance of each algorithm. Each optimizer is then implemented within a Neural Network (NN) framework as an alternative to the conventional backpropagation (BP) optimization. The Wine Quality dataset, sourced from the Kaggle repository, is used to evaluate the robustness of each optimizer. This dataset serves as a suitable testing ground for comparing the performance of these algorithms, providing insights into their strengths and limitations in handling real-world data scenarios.

A. Initial Hypothesis:

The general hypothesis (H) of the study is as follows;

- *Random Hill Climbing:* RHC will have the lowest performance among the three optimizers due to its inability to escape local minima with large basins of attraction [4]. It will struggle to find the global optimum in the KC problem due to its stochastic search method, which may not effectively handle the structural complexities of graph solutions. However, RHC will perform adequately in neural network training scenarios, especially in quick evaluation times, making it the fastest optimizer for all problems.
- *Simulated Annealing:* SA will outperform both GA and RHC on FP by effectively exploring and escaping local optima through its probabilistic acceptance of worse solutions early in the search. On KC, SA's performance will be comparable to RHC due to the structural complexity of graph solutions. In neural network training, SA will outperform RHC and GA because of its ability to quickly escape local minima, resulting in better convergence and a reduced risk of overfitting.
- *Genetic Algorithm:* GA will perform better than RHC but worse than SA on FP due to the higher cost of function evaluations despite utilizing crossover operations to explore diverse regions of the search space. GA will outperform KC and produce more reliable global optima by efficiently managing structural alterations in graph solutions. However, because of its numerous parameters and high computing overhead, GA might not be as feasible for direct substitution in backpropagation for neural networks as faster optimizers like RHC and SA.

II. Random Optimizers

Random optimization is a category of algorithms that use probabilistic methods to explore the solution space. These methods are highly successful for complex landscapes with many local optima, as they use randomness to escape these minima and locate the global optimum. The stochastic nature of these algorithms allows for a more flexible exploration of the solution space compared to deterministic methods.

Three methods are commonly explored from this class of optimizers: Random Hill Climbing, Simulated Annealing, and Genetic Algorithms. These methods vary in complexity and how they implement randomness to solve optimization problems effectively.

A. Random Hill Climbing:

This algorithm starts from a random solution and iteratively moves to neighbouring solutions, adopting better ones. The process continues until no better neighbours are found, with randomness enabling the exploration of different solution space regions [7]. Critical parameters are:

- *Step Size:* Controls the magnitude of random changes to the current solution. Too small limits exploration, while too large can miss local exploitation.
- *Number of Restarts:* Specifies how often the algorithm restarts from a new random initial solution, helping avoid local optima by giving multiple chances to find a global optimum.

B. Simulated Annealing:

Inspired by the metallurgical annealing process, this algorithm simulates cooling to reach a minimum energy state. It starts with a high "temperature" for broad exploration, gradually decreasing to favour improvements while allowing occasional worse moves to escape local optima [7]. Key parameters are:

- *Initial Temperature:* Sets the starting temperature, with a higher value facilitating broader exploration and a lower value focusing on exploiting the best solutions.
- *Decay Rate:* Controls how quickly the temperature decreases, balancing thorough exploration with convergence speed.

C. Genetic Algorithm:

Mimicking natural selection, genetic algorithms evolve a population of solutions over generations. A fitness function evaluates solutions, and the best performers are selected for crossover and mutation to create a new population for each generation [7]. Key parameters are:

- *Population Size:* Determines the number of solutions in each generation. A larger size increases diversity and exploration but requires more computational resources, while a smaller size speeds up the algorithm but reduces diversity.
- *Mutation Rate:* Defines the probability of random individual changes. A higher rate helps escape local optima but can disrupt convergence if excessive; a lower rate preserves good solutions but may reduce exploration.

In optimization algorithms like RHC, SA, and GA, the *Max Attempts* and *Max Iterations* are critical convergence parameters [2]. RHC uses them to limit iterations without fitness improvement and cap total iterations. SA adds a temperature schedule to manage the acceptance of worse solutions, balancing exploration with convergence. GA's evolve populations through selection, crossover, and mutation across generations. Despite these variations, all aim to balance exploration and exploitation, stopping when a satisfactory solution is found or computational limits are reached, ensuring effective optimization within set constraints.

III. Optimization Problems

Two discrete benchmark problems are explored to evaluate the performance of the random optimization algorithms: the *Four Peaks* problem and the *Max k-Colour* problem. Both problems are formulated as maximization problems to allow for one-to-one comparison and analysis of each algorithm's overall performance. Each problem is explored using the *mlrose* [2] python package.

A. FourPeaks

The *Four Peaks* problem is a well-known benchmark in optimization and is particularly useful for evaluating the balance between exploration and exploitation in algorithms. The problem is defined on a binary string of length n. The fitness function, f, is designed to reward strings that have a significant number of consecutive 0s at the beginning and consecutive 1s at the end. Formally, the fitness function is defined as:

$$
\begin{aligned}
f(x,T) &= \max(tail(0,x), head(1,x)) + R(x,T) \qquad (1)\\
tail(b,x) &= \textit{is the number of trailing } b\textit{'s in } x\\
head(b,x) &= \textit{is the number of leading } b\textit{'s in } x\\
R(x,T) &= \begin{cases} n, & \text{if } \ tail(0,x) > T \text{ and } head(1,x) > T \\ 0, & \text{otherwise} \end{cases}
\end{aligned}
$$

T is a threshold parameter expressed as a percentage of the state space dimension, and R is an additional reward applied when both *tail* and *head* are greater than T. This problem has two local maxima and two global maxima, which becomes increasingly difficult to identify for large values of T because the basin of attraction for the inferior local maxima becomes larger [4]. Therefore, it becomes an interesting challenge for random optimizers as it highlights the challenges of escaping local optima in search of small global peaks.

B. Max k-Colour

The *Max k-Colour* problem is a combinatorial optimization problem that involves colouring the nodes of a graph such that no two adjacent nodes share the same colour, using at most k colours. The fitness function for this problem is defined as:

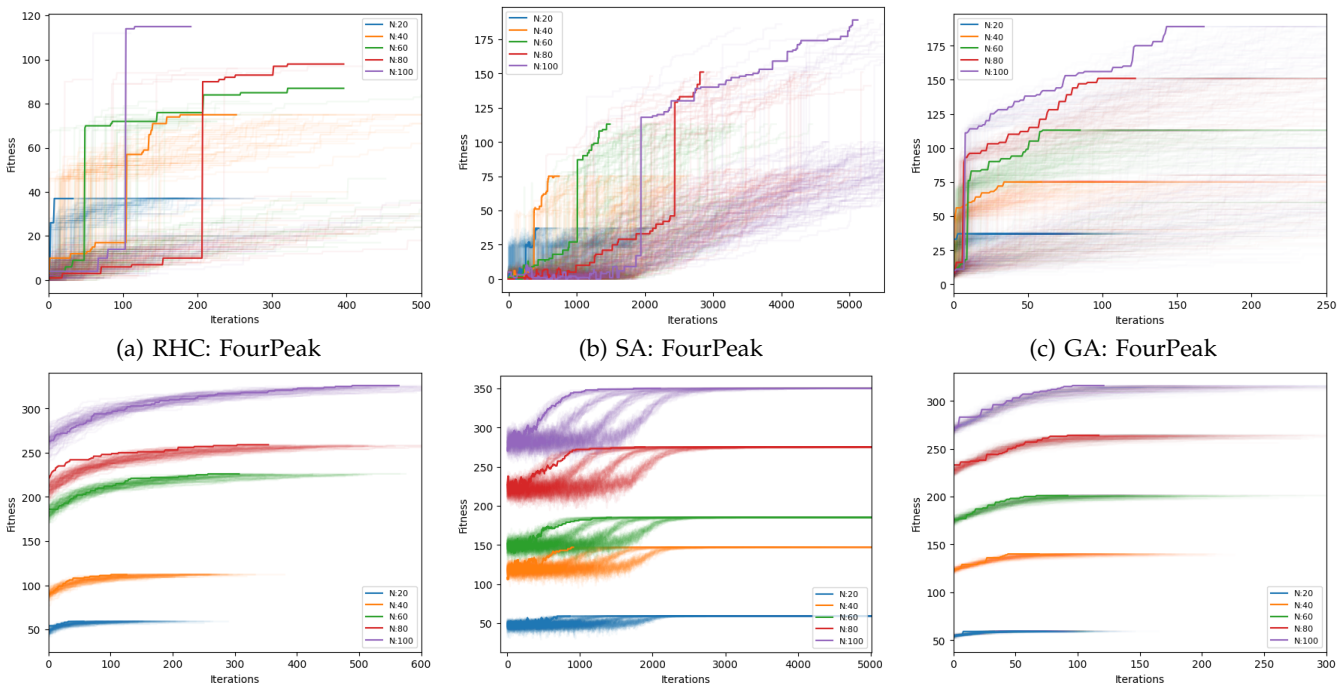
$$f(x) = \#\ \textit{edges with different-coloured endpoints} \qquad (2)$$

The goal is to maximize this fitness function, effectively minimizing the number of conflicts (i.e., edges with same-coloured endpoints). This problem presents a very interesting challenge as it is known to be NP-Complete [4], thus making it an ideal case for algorithm comparison. Additionally, the problem contains an underlying structure that may present challenges for random memoryless optimizers.

IV. Optimizer Analysis

The analysis of the three random optimizers, RHC, SA, and GA, begins with systematically investigating their performance on the two academic benchmark problems: the *Four Peaks* and the *Max K-Colour* problem. This investigation aims to gain insight into each optimizer's overall effectiveness by considering several key aspects, such as problem complexity, number of function evaluations, number of iterations, and overall evaluation time.

The optimization process is repeated with five random seeds to ensure adequate opportunity for randomized exploration. Additionally, the performance of each optimizer is analyzed across different hyper-parameter settings to understand their impacts on the optimization outcomes. A summary of the corresponding explored parameters is summarized in Table I. This includes variables specific to the explored problem definitions and the random optimization algorithms.



(a) RHC: FourPeak (b) SA: FourPeak (c) GA: FourPeak

(d) RHC: k-Colour (e) SA: k-Colour (f) GA: k-Colour

Fig. 1: Optimizer Complexity Performance Curves

TABLE I: Optimizer parameter summary

Hyper-parameter	Value
Problem Definition	
Fourpeak- string size	{20, 40, 60, 80, 100}
Fourpeak- threshold	{0.20}
k-Colour - node number	{20, 40, 60, 80, 100}
k-Colour - max adjacent edges	{6}
k-Colour - max number colors	{5}
Algorithm Constants	
Maximum Attempts	{25, 50, 75}
Maximum Iterations	{50,000}
RHC	
Restart size	{0, 25, 50, 100}
SA	
Initial Temperature	{10, 100, 1000}
Decay rate	{0.99}
GA	
Population size	{25, 50, 100}
Mutation size	{0.2, 0.5, 0.8}

V. Optimizer Complexity Curves

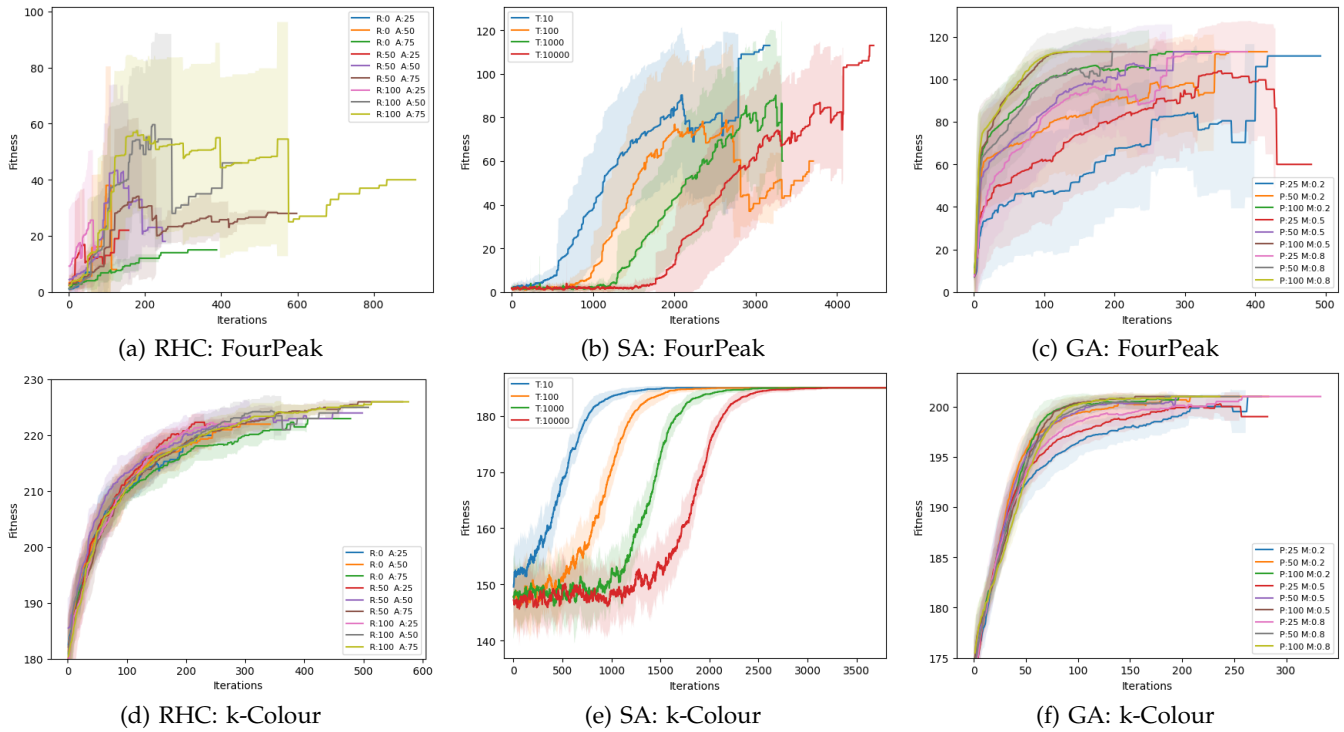
This section examines how each optimizer's performance varies with different levels of problem complexity. For the *Four Peaks* problem, the algorithms are evaluated as the size of the binary string varies. For the *Max K-Colour* problem, the number of nodes in the graph is varied. By systematically varying the string sizes (n_s) and the number of nodes (n_o), the investigation provides a comprehensive view of each optimizer's scalability and robustness to problem complexity. Additional parameters, such as the threshold factor, number of colours, and number of adjacent edges, were fixed for this study due to computational and time resource limitations. The complexity curves for each optimizer on the *Four Peaks* and *Max K-Colour* problems are analyzed in Figure 1.

A. Problem 1: Four Peaks:

1) *Random Hill Climbing:* As problem complexity increases, the average number of iterations increases, improving overall fitness. However, global maxima are reached only for smaller string lengths (20 and 40). More complex problems often find only local optima, requiring additional attempts and restarts, which increase computational demand.
2) *Simulated Annealing:* With increasing complexity, the number of iterations grows. The noisy fitness curve due to initial high temperature stabilizes as the temperature decreases, leading to consistent improvements. Optimal fitness is identified for each problem size, but high temperatures can delay convergence, significantly increasing the number of function evaluations.
3) *Genetic Algorithm:* GA performs well, finding global optima for every problem size. It converges faster than other algorithms due to its parallel nature, although this increases computational demand. Consistent convergence plateaus suggest that the population size is sufficient to locate the global optimum within a consistent number of iterations.

B. Problem 2: Max K-Colours:

1) *Random Hill Climbing:* As the number of nodes increases, solutions form discrete clusters. While performance improves with more attempts, the computational time increases. While the optimizer often finds the global optimum only for the smallest



(a) RHC: FourPeak (b) SA: FourPeak (c) GA: FourPeak

(d) RHC: k-Colour (e) SA: k-Colour (f) GA: k-Colour

Fig. 2: Hyper-Parameter Optimizer Curves

problem size, RHC performs adequately within a reasonable time for the remaining sizes.

2) *Simulated Annealing:* SA consistently converges to a solution for all complexity levels. High temperatures increase the likelihood of selecting worse solutions early on, leading to higher computational demand. The algorithm converges when the temperature becomes sufficiently low, but many iterations reach the maximum limits without better solutions. Unfortunately, this observed phenomenon is a quirk of the SA convergence criteria when temperatures are extremely large, thus leading to repeated restarts and unnecessary function evaluations.
3) *Genetic Algorithm:* GA performs well on complex problems, though it heavily relies on tuning parameters. The algorithm is typically robust in combinatorial problems, as the crossover operator combines good solutions to create better offspring, thus naturally retaining the problem structure. Maintaining a diverse population aids in the effective exploration of the solution space. GA converges with fewer iterations and more consistently than Simulated Annealing, though a larger population size could yield even better results.

VI. Optimizer Individual Performance Curves

We explore the specific algorithm's parameter sensitivity further. In the context of each particular problem, a fixed complexity size ($n = 60$) is established to allow for a robust comparison. Such an analysis is crucial for understanding the strengths and weaknesses of each optimization algorithm, particularly in handling complex and large-scale problems. Figure 2 presents the impact of the varying parameters for each optimizer on the *Four Peaks* and *Max k-Colour* problems. Table V shows the mean performance summary of all analyzed metrics.

TABLE II: Optimizer Results summary

	RHC	SA	GA
FourPeaks (n = 60)			
max $f(x)$	115.0	189.0	189.0
$f(x)$	13.08 ± 14.78	109.22 ± 33.44	104.99 ± 62.38
Iters (10^3)	0.13 ± 0.11	5.71 ± 0.95	0.30 ± 0.20
FEval (10^2)	21.73 ± 29.41	44.91 ± 12.79	107.46 ± 61.91
time	0.04 ± 0.05	0.50 ± 0.17	2.84 ± 2.75
Max k-Colour (n = 60)			
max $f(x)$	226.0	185.0	201.0
$f(x)$	222.21 ± 6.23	184.89 ± 0.56	199.77 ± 1.86
Iters (10^3)	0.30 ± 0.11	31.99 ± 21.65	0.15 ± 0.05
FEval (10^2)	49.72 ± 57.61	394.66 ± 261.21	88.42 ± 49.12
time	2.09 ± 2.01	5.16 ± 3.44	1.82 ± 1.03

A. Problem 1: Four Peaks:

1) *Random Hill Climbing:* Increasing the number of restarts generally improves fitness. Specifically, 100 restarts with 50 or 75 attempts consistently yield better solutions, though they do not guarantee optimal outcomes due to stochastic evaluation and varying seeds. This effect is observed in the large uncertainty bounds and erratic means.
2) *Simulated Annealing:* The temperature significantly impacts the SA algorithm, demonstrating considerable sensitivity to the initial starting point. The lower temperature, $10^{\circ}C$, accelerates finding the optimal region. High temperatures prolong conver-

gence, increasing computation costs. Constant cooling schedules exhibit consistent convergence slopes, with the potential for faster convergence through optimized cooling schedules.

3) *Genetic Algorithm:* Population size significantly impacts performance; a size of 100 appears highly effective. However, the mutation rate impact is less clear, but a rate of 0.50 shows quick convergence with narrower uncertainty. Therefore, the problem definition favours a slightly more stochastic approach with a larger population size.

B. Problem 2: Max k-Colour:

1) *Random Hill Climbing:* Performance variations with restarts and attempts are minimal. Increased attempts improve performance but at increased computational cost. The current problem structure likely contains many local minima of similar magnitude. Therefore, random exploration effectively finds reasonably good solutions in this problem's structure.
2) *Simulated Annealing:* Again, lower initial temperatures facilitate quicker convergence. Interestingly, all temperatures converge to the same optimal region. As witnessed before, high temperatures risk delayed convergence without identifying better solutions, requiring careful temperature management and selection to avoid unnecessary evaluations.
3) *Genetic Algorithm:* Similar to RHC, optimal parameter differentiation is less clear. Again, a population size of 100 consistently achieves faster convergence. However, in this problem, a lower mutation rate is favourable, indicating the problem's structured nature favours GA crossover methodologies over a more random beam searching approach.

VII. Optimizer Efficiency Comparison

After evaluating the general complexity and individual performance characteristics, a general efficiency performance comparison is conducted for each optimizer. To determine which optimizer performs most efficiently for each problem, a detailed analysis of inner function evaluations and wall-clock time is conducted. Figure 3 summarizes the results for each problem across all explored problem sizes. Two key metrics are considered: pure evaluation time and time-normalized function evaluations. These metrics provide a good understanding of solution complexity and overall optimizer efficiency. Note that these metrics are averaged across all seeded evaluations.

For the *Four Peaks* problem, both RHC and SA showed similar function evaluations per second. However, GA has a much lower rate. This does not translate to faster evaluation times, as GA is approximately ten times slower than the other two optimizers. In the *Max k-Colour* problem, SA is the least efficient optimizer. The number of function evaluations for SA is significantly higher than for RHC and GA, resulting in a severe penalty in evaluation time. In this particular problem, GA proves to be the most efficient. Possible reasons for this efficiency difference relate to the specific problem characteristics.

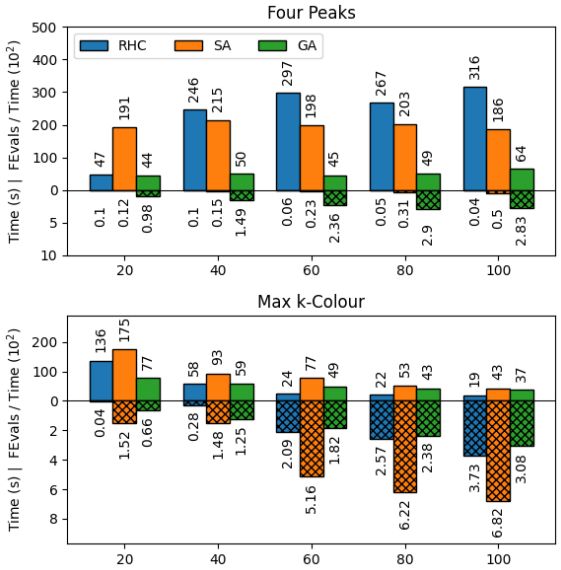


Fig. 3: Normalized function evaluation and wall-clock time comparison for varying complexity

In the Four Peaks problem, the potentially large population size and relatively simple search space made GA less effective compared to the more randomized RHC. For the *Max k-Colour* problem, the presence of many local optima meant that while SA could quickly find improvements, it eventually became stuck, leading to excessive iterations. Conversely, GA was able to capture the problem structure better and slowly evolve towards a good solution.

VIII. Neural Network

Having investigated the effectiveness of various optimizers on discrete academic benchmarks, the focus shifts to evaluating random optimization algorithms for tuning neural network weights, replacing the conventional backpropagation (BP) algorithm. This study explores whether RHC, SA, and GA can be possible alternatives to BP for training neural networks.

A. Dataset Introduction

The performance of these optimizers is assessed using the Wine Quality dataset, a challenging real-world source from the Kaggle repository. This dataset consists of 1,143 instances and 11 features related to the Portuguese 'Vinho Verde' wine. It focuses on classifying wine quality based on physico-chemical properties such as acidity and pH levels, with quality scores ranging from 3 (poor) to 8 (excellent) [6]. This dataset presents notable challenges for model comparison due to its multi-class nature with ordered quality scores and significant class imbalance. It requires specialized evaluation metrics beyond accuracy. Additionally, the subjective nature of wine classification introduces a high degree of noise into the metric, making the problem even more

challenging to converge to a global optimal solution as many local optima are present.

B. General Methodology

This study employs a methodology similar to that demonstrated in [1], implementing a hybrid training strategy that combines data standardization, cross-validation methodologies, and hold-out test evaluations to ensure a reliable assessment of each algorithm's performance. Given the extreme imbalance characteristics of the dataset, a stratified sampling approach is employed throughout the training and cross-validation procedures to preserve class distribution within the testing sets. The dataset target distributions for the train and test splits are shown in Figure 4.

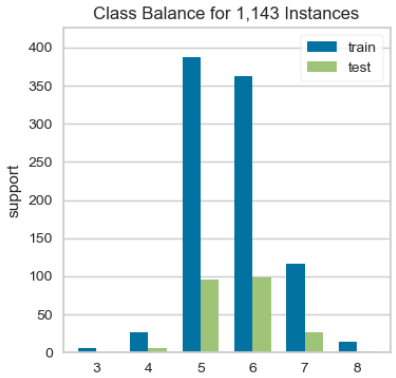


Fig. 4: Class distribution between train (blue) and test (green) datasets

To ensure a fair comparison between the various optimizers, the optimal neural network model previously determined using backpropagation from study [1] is used as a benchmark. However, the analysis is conducted with a different Python library, PyPerch [3], as it contains the methods necessary to apply random optimizers and backpropagation concurrently. A 5-fold cross-validation approach is applied within the training set to maintain consistency and comparability to re-optimize hyper-parameters for optimal backpropagation performance. The baseline modelling parameters are then fixed for the remaining methods. A secondary grid-search evaluation is conducted to explore and evaluate the key hyper-parameters of the various optimizers based on the previous learnings. Parameter configurations vary depending on the algorithm, as summarized in Table III. It should be noted that other parameters and ranges could be tuned. However, not all configurations and parameters could be optimized due to the limited time and computing resources.

IX. Neural Network Analysis

The optimal hyper-parameter results for the baseline model and associated random optimizers are summarized in Table IV. Comparing these results with [1] indicates slight variations in network structure, though activation, learning rate, and optimization algorithms remain consistent. These differences may stem from initial sampling or slight code variations. However, the close overlap provides confidence in a fair comparison of the Wine dataset.

TABLE III: Summary of grid search hyper-parameters

Hyperparameter	Value
BP	
Learning Rate	{0.1, 0.2}
Layer Size	{2, 3}
Number Nodes	{10, 20}
Epochs	{100, 1000, 3000}
Activation Functions	{Tanh, Relu}
Optimizer	{Adam, SGD}
RHC	
Step size	{0.1, 0.2, 0.3}
SA	
Step size	{0.1, 0.2}
Initial Temperature	{100, 1000, 10,000}
Decay rate	{0.90, 0.99}
GA	
Population size	{50, 100, 200}
Mutation size	{10, 25, 50}
Mate size	{30}

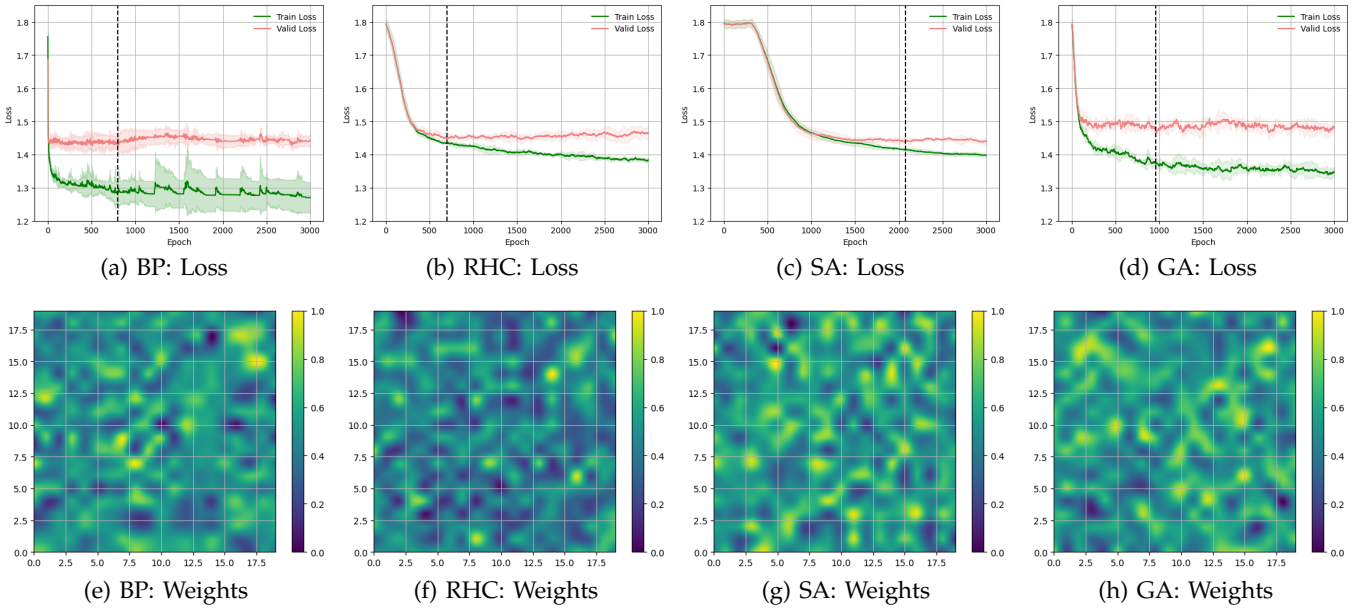
TABLE IV: Grid search hyper-parameter results

Baseline (BP) Neural Network	
{Node (Layer): [20, 20](2), Acti: Tanh, Epoch: 3000, Opti: Adam}	
Random Optimizers	
RHC:	{Step size: 0.2}
SA:	{Step size: 0.1, Initial Temp: 10,000°C, Decay rate: 0.99}
GA:	{Population size: 100, Mutation size: 50, Mate size: 30}

X. Neural Network Loss Curve

Analyzing internal training loss directly is crucial for evaluating optimizer performance, highlighting optimal epochs for early stopping and achieving low validation errors. Additionally, comparing loss curves is valuable to determine if different optimizers reach similar convergence levels. Figures Figs. 5a–5d illustrate training and validation loss curves, with validation minima marked by black dashes.

- BP achieves the lowest training loss, indicating its effectiveness in minimizing errors during training. However, the irregularities observed suggest possible challenges in stabilizing convergence, hinting at opportunities for fine-tuning learning rates and regularization techniques to avoid fluctuations and potentially improve performance further.
- RHC shows smoother loss curves compared to BP but begins to overfit around 700 epochs, where the validation loss diverges from the training loss. This early overfitting suggests limitations in its ability to generalize to unseen data without further adjustments or regularization.
- SA demonstrates robust performance, maintaining low validation loss up to approximately 2075 epochs. The gradual increase in validation loss beyond this point indicates the optimizer's resilience to overfitting over extended training periods, suggesting that continued training could yield further improvements in model generalization.



(a) BP: Loss (b) RHC: Loss (c) SA: Loss (d) GA: Loss

(e) BP: Weights (f) RHC: Weights (g) SA: Weights (h) GA: Weights

Fig. 5: Neural Network Epoch Loss Curves and output weights for varying optimizers

- In contrast, GA shows the highest validation loss early in training, indicating sub-optimal convergence compared to other optimizers. The rapid increase in validation loss suggests early overfitting tendencies, highlighting the need for revisiting hyper-parameter settings or exploring alternative optimization strategies to the specific characteristics of the problem domain.

These findings show the importance of selecting an appropriate optimizer tailored to the neural network architecture and dataset characteristics. BP's low training loss despite irregularities suggests potential for further refinement. RHC and GA indicate signs of early overfitting, thus requiring careful parameter tuning to enhance generalization. SA's sustained performance over extended epochs indicates its effectiveness in prolonged training scenarios.

A. Neural Network Weight Analysis

Examining optimized internal neural network weights at the output layer provides another valuable comparison metric. Figure Figs. 5e–5h show distinct weight distributions for each optimizer, indicating different minima despite similar obtained loss values for most optimizers.

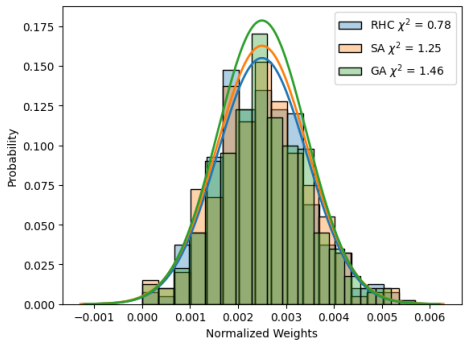


Fig. 6: Neural Network weight distributions comparison

To quantify the degree of similarity between these weights, the chi-squared (χ^2) distance metric [9] is employed. This statistical measure is used to compare the dissimilarity between two probability distributions. Lower values of χ^2 indicate more similar distributions. Using BP as a baseline, the scores were obtained for each optimizer, summarized in Figure 6. The following key takeaways are observed:

- The RHC optimizer exhibits the smallest distance ($\chi^2 = 0.78$) compared to the BP algorithm. This indicates that RHC produces weight distributions most similar to those of BP, suggesting that it finds a similar minimum in the optimization landscape.
- Both SA and GA have higher distances ($\chi^2 = 1.25$ and $\chi^2 = 1.46$, respectively) compared to BP. This implies that while their weight distributions are somewhat similar to each other, they differ more significantly from those produced by BP.

Despite the differences in the chi-squared distances, the overlap of the weight distributions for all optimizers is notable. This overlap indicates that all optimizers find solutions within a similar range of the weight space, even though the exact distributions vary, thus indicating moderately similar convergence.

B. Discrete versus Continuous Weights

The weights in a neural network are continuous and real-valued, requiring special considerations when applying random optimization algorithms. Unlike discrete domains, the continuous nature of neural network weights results in a much larger real-valued solution space. BP efficiently updates weights using gradient information, making it highly effective for smooth, differentiable loss surfaces. Conversely, random optimization algorithms do not use gradients and rely on stochastic processes, which makes them robust in non-differentiable or irregu-

lar landscapes but typically slower to converge. They can better escape local minima through probabilistic acceptance of worse solutions (SA) and crossover/mutation (GA). BP's efficiency and speed often make it preferable for large datasets and complex models. However, random optimization algorithms are valuable in scenarios where gradients are not well-defined (avoids vanishing gradients [10]) or the loss landscape is highly irregular.

XI. Neural Network Optimizer Comparison

Table V presents the final test set evaluated results for the Wine Dataset using BP, RHC, SA, and GA. The performance metrics considered are Accuracy, Precision, Recall, F1-score, the number of Epochs, and Training Time (in seconds).

TABLE V: Final test set evaluated results

	Wine Dataset			
Metrics	BP	RHC	SA	GA
Accuracy	0.651	0.650	0.651	0.611
Precision	0.620	0.618	0.616	0.580
Recall	0.651	0.651	0.651	0.611
F1-score	0.635	0.630	0.633	0.591
Epochs	800	700	2075	960
Train Time (s)	48.96	77.87	213.23	2472.60
Epoch/Time	16.33	9.00	9.73	0.40

The following general outcomes are observed based on the overall performance:

- BP demonstrates the highest precision and F1-score while matching SA and RHC in accuracy and recall. It also has the fastest training time, making it the most efficient optimizer for the Wine Dataset due to its direct gradient-based optimization approach.
- RHC performs similarly to BP in accuracy and recall but lags in precision and F1-score. It requires more training time than BP but fewer epochs, indicating longer epochs per session. RHC is cost-effective for simpler problem landscapes but may struggle with more complex ones due to its stochastic nature.
- SA achieves accuracy and recall comparable to BP and RHC but has slightly lower precision and F1-score. It requires significantly more training time and epochs however function evaluation time is similar to RHC's. SA is suitable for problems with multiple local minima but needs careful tuning of the cooling schedule for optimal performance.
- GA shows the worst performance across all metrics, with the lowest accuracy, precision, recall, and F1-score. It has the longest training time, highlighting its low efficiency and high cost. The GA effectiveness depends heavily on parameter tuning, making it less suitable for the Wine Dataset in its current configuration due to high computing demands.

XII. Conclusion

In conclusion, not only does the performance of each optimizer vary, but their efficiency can also drastically change depending on the particular problem. This observation aligns with the no-free-lunch theorem [8], which suggests that no single optimizer is best for all problems. A general optimizer summary and hypothesis confirmation (✓) or rejection (×) is indicated in Table VI.

TABLE VI: Model result and hypothesis summary

Models	Pros	Cons
RHC	✓ Fastest optimizer in terms of wall-clock times. • Minimal parameter tuning required. × Effective for small or moderately sized problems. (Ⱨ: Achieves better prediction performance on NN problem than GA)	✓ Prone to getting stuck in local minima. ✓ Not suitable for complex and structured problems. • Stochastic nature means no guarantee of finding the global optimum.
SA	✓ Escapes local optima by accepting worse solutions initially. ✓ Performs well on problems with many local minima (FP and NN). • Highly flexible with a tunable temperature and cooling schedule (Ⱨ: worse than MLP on binary class).	× Computationally expensive with many iterations. (Ⱨ: Worse performance on structured problems than RHC and GA. • Sensitive to cooling schedule; requires careful tuning. • Slower overall convergence compared to RHC and BP.
GA	× Robust for large complex problems with many local optima. (Ⱨ: Lowest NN prediction performance) • Population diversity helps explore the space effectively. × Capable of capturing structural patterns over iterations (Ⱨ: Lower prediction performance than RHC)	✓ Most computationally intensive with the longest training time. • Requires significant parameter tuning for population size and mutation rate.

This analysis demonstrates the effectiveness of random search approaches in navigating non-differentiable, discontinuous, or irregular optimization landscapes. However, it is crucial to appropriately tune individual optimizers and compare multiple options to identify the optimal performer for each problem. By carefully selecting and tuning optimizers, we can achieve better solutions tailored to each problem's unique characteristics.

XIII. Resources

[1] *CS7641 A1: Supervised Learning.* Odendaal, K. (2024).
[2] *API Reference.* ml-rose-hiive. Rollings, A., Hays, G. https://github.com/hiive/mlrose.
[3] *API Reference.* pyperch. Mansfield, J., https://github.com/jlm429/pyperch.
[4] *MIMIC: Finding optima by estimating probability densities*. De Bonet, J., Isbell, C., Viola, P. (1996).
[5] *Machine Learning LaTeX Template*. Nakamura, K. (2023).
[6] *Wine Quality Dataset*. Kaggle. https://www.kaggle.com/datasets/yasserh/wine-quality-dataset.
[7] *Clever Algorithms: Nature-Inspired Programming Recipes.* Brownlee, J. 1st ed, (2012).
[8] *Machine Learning*. Mitchell, T. M. vol. 1, (1997).
[9] *Hands-on machine learning with Scikit-Learn, Keras and TensorFlow: concepts, tools, and techniques to build intelligent systems.* Geron, A. 2nd ed, (2019).
[10] *Artificial Intelligence: A modern approach*. Russel, S., Norivg, P. 4th ed, (2021).