CS7641 Assignment 2 - Randomized Optimization

Melody Yu
CS7641: Machine Learning, Summer 2024
Georgia Institute of Technology
myu352@gatech.edu

***Abstract*—This project explores the application of randomized optimization (RO) algorithms to solve discrete optimization problems and compare neural network performance. We first apply three RO algorithms –Randomized Hill Climbing (RHC), Simulated Annealing (SA) and Genetic Algorithm (GA)– to two optimization problems, Max K-Colors and Four Peaks. We then use the Abalone dataset to compare neural network weights obtained with traditional backpropagation with those obtained through these algorithms. This report details our hypotheses and subsequent analysis of the impact of tuning various hyperparameters on algorithm convergence and exploring the solution space.**

I. Introduction

In the realm of optimization methods, random optimization (RO) search algorithms offer a contrast to deterministic methods. While a random search may not intuitively offer a better solution, their randomness makes them more robust to noise and uncertainties in the data, as they do not rely on precise gradient information, making them suitable for noisy environments where exact objective values are difficult to obtain. [1]. In this project, we aim to explore the extent of their robustness and their underlying convergence criteria by applying RO algorithms in several contexts.

We begin by selecting two discrete optimization problems: Four Peaks and Max K Colors. Each optimization problem domain is adjusted such that their fitness function is maximized rather than minimized, achieved by negating the loss function. We then implement three RO algorithms: Randomized Hill Climbing (RHC), Simulated Annealing (SA), and Genetic Algorithm (GA).

The Four Peaks problem is defined on a bit string of length n, where the objective is to find strings with significant leading and trailing ones, separated by zeros. In this case, a higher fitness value refers to strings that have long runs of consecutive 1's and 0's at both ends of the string. This problem introduces a deceptive landscape that can mislead optimization algorithms, making it excellent for evaluating SA. SA highlights the Four Peaks problem effectively because it can escape local optima through probabilistic hill climbing. The gradual cooling schedule of SA allows it to accept worse solutions temporarily, providing a mechanism to escape local optima and explore the search space more broadly, ultimately finding better global solutions. SA's strength in avoiding local optima and exploring the search space globally will be showcased in later sections, while also highlighting the algorithm's dependency on carefully tuned parameters such as cooling schedules and temperature.

The Max K-Colors problem involves coloring the nodes of a graph with K colors such that no two adjacent nodes share the same color. A high fitness value is equivalent to minimal conflicts as each node's color would differ from their neighbor. This problem is interesting because it represents a classic combinatorial optimization challenge with a large search space which is ideal for testing the capabilities of GA. The GA's ability to explore large and complex search spaces through crossover and mutation operations is highlighted, along with its susceptibility to getting stuck in local optima if not properly tuned.

Additionally, we extend the application of RO algorithms to a neural network (NN) using the Abalone dataset [2]. By utilizing the PyPerch [3]library, we explore the effectiveness of RO algorithms in a continuous problem space, substituting traditional backpropagation with RO methods. This approach enables us to assess the robustness and applicability of RO algorithms beyond discrete optimization, highlighting their potential in various domains.

The Abalone dataset focuses on predicting the age of 4177 instances of abalones based on seven physical measurements. For this analysis, we performed a binary classification task to determine whether an abalone has 11 rings or more based on physical features, such as length and weight. This dataset is particularly interesting due to its multicollinearity, which can heavily affect algorithm performance and convergence by increasing the risk of overfitting. Considering the class imbalances, we opt to use F1-score as our performance metric.

The sections below are organized by optimization problem domain and further split by RO algorithm. We begin each section with a hypothesis about the expected performance of the algorithm's hyperparameter(s), describe our experimental procedure and analyze hyperparameters, results, and performance.

While we discuss convergence in each section, for the Four Peaks section and Max K-Colors section, we particularly highlight the performance of simulated annealing and genetic algorithms, respectively.

II. Four Peaks

We hypothesize that Simulated Annealing (SA) will outperform Randomized Hill Climbing (RHC) on the Four Peaks problem regardless of problem size in terms of function evaluations (Fevals). This is because SA's gradual cooling schedule allows it to accept worse solutions temporarily, enhancing its ability to escape local optima and explore the search space

more thoroughly compared to RHC, which often gets stuck in local optima due to its purely greedy nature [4]). Similarly, while Genetic Algorithms (GA) can explore multiple solutions simultaneously due to their population diversity, they can suffer from premature convergence, especially in deceptive landscapes like Four Peaks, where the global optimum is not directly aligned with incremental improvements. [4]

Though the hyperparameters will vary between algorithms, each algorithm generally follows the same format. With the mlrose-hiive library [5], we use the generator class to define three Four Peaks problems of sizes 25, 50, and 100, to examine differences in algorithm performance within large, medium and small search spaces. When tuning hyperparameters, we ensure a fair comparison by locking other parameter values such as seed, number of iterations, and max attempts (arbitrarily decided on by taking runtime into account) so that the hyperparameter is the only changing variable. We do a broad analysis on the hyperparameter by looking at a wide range of values for the hyperparameter and perform 5-fold cross validation to determine its effect on the problem and calculate the 'best' performing in terms of highest fitness and highest FEvals, defined as the number of evaluations per iteration. We retain those hyperparameters for after tuning, when we average the performance of the algorithm with those tuned features across three seeds and compare results against each other.

A. Simulated Annealing

For Simulated Annealing (SA), we choose to focus on temperature, as it determines the probability of accepting worse solutions at the start. We hypothesize that smaller problem sizes would benefit more from higher initial temperatures since they are likely to frequently land in local optima. However, we also expect high temperatures across all problem sizes to help escape local optima quickly and maximize fitness.

Fig. 1 shows that we were partly correct. Higher temperatures generally reached higher fitness levels quickly, regardless of problem size. All temperatures converged at different iterations for the various problem sizes, likely due to the ease of escaping local optima increasing with higher temperatures. This suggests that the trade-off between exploration and exploitation is balanced such that the model can converge to a solution by increasing the temperature hyperparameter. Interestingly, the problem size of 50 achieved a higher fitness on average compared to the problem size of 100. This can be attributed to the increased search space complexity with larger problem sizes, making it more challenging to find the global optimum and resulting in lower fitness scores.

We also examine decay rates across problem sizes since decay type affects the algorithm's ability to escape local optima and consequently achieve high fitness values. We anticipated that geometric decay would be more impactful because it gradually reduces temperature compared to exponential decay, which reduces temperature faster. However, for smaller problem sizes, we observed that exponential decay was more effective. We attribute this to the fact that smaller problem sizes require fewer iterations to reach high fitness scores and near-optimal solutions, making exponential decay's quicker cooling advantageous.

Problem Size	Temperature	Decay
25	750	Exponential
50	750	Geometric
100	75	Geometric

TABLE I
SIMULATED ANNEALING HYPERPARAMETERS FOR FOUR PEAKS

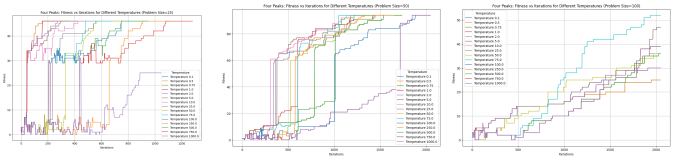

Fig. 1. Temperature (0.1-1000) effects on fitness and convergence for different problem sizes.

In Fig. 2, we observe that SA is more efficient at exploring the search space compared to GA and RHC. While GA may achieve the highest fitness, it does so at the expense of significantly higher function evaluations (FEvals), making it computationally expensive. This is due to GA's need to evaluate multiple individuals in each generation, along with performing crossover and mutation operations, to assess the fitness of the population.

In contrast, SA's single-solution approach leads to fewer evaluations and lower computational cost. SA adjusts its solution gradually, guided by the temperature schedule, which allows it to explore the search space efficiently without the overhead of managing a population. This efficiency is evident in the relatively low number of FEvals required by SA. Additionally, because of its single-solution approach, seen in Fig. 3, SA takes less time on average the GA (Fig. 4) and RHC (Fig. 5) across problem sizes.

RHC, while having lower FEvals compared to GA, starts from a single point and makes incremental changes to improve fitness. This method results in fewer evaluations than GA but still underperforms compared to SA due to its tendency to get trapped in local optima, leading to lower fitness levels.

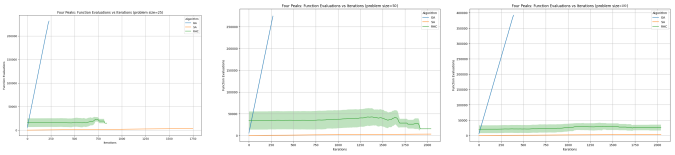

Fig. 2. Plotting FEvals/Iterations shows that SA has a better trade-off between computational cost and optimization performance across different problem sizes.

B. Genetic Algorithm

For genetic algorithms, we examine population size and mutation rate. We hypothesize that a larger population size will benefit fitness by offering a more diverse set of solutions, thereby avoiding premature convergence to local optima. Additionally, we anticipate that Four Peaks would not benefit from

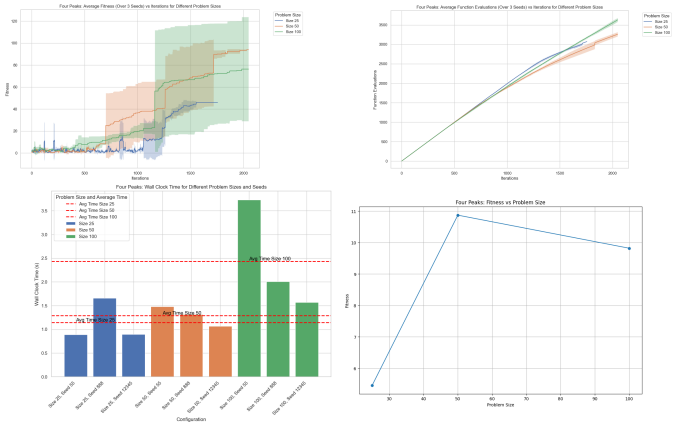

Fig. 3. Plots of fitness/iteration, fitness/problem size, FEvals/iterations and wall clock time for SA problem sizes 25, 50 and 100 averaged across three seeds, using the hyperparameters that returned highest fitness.

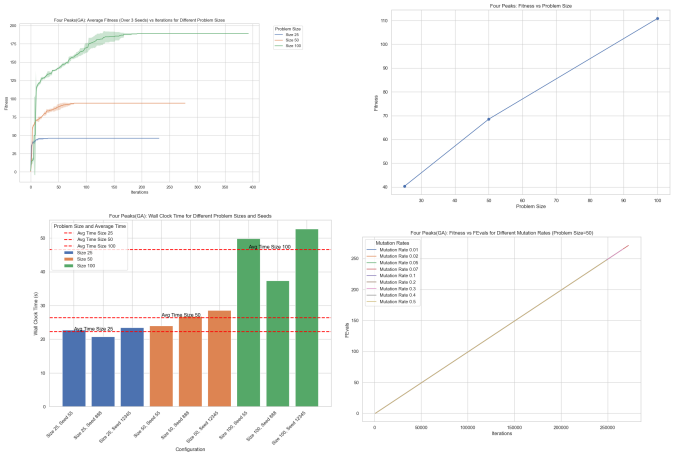

Fig. 4. Plots of fitness/iteration, fitness/problem size, FEvals/iterations and wall clock time for GA problem sizes 25, 50 and 100 averaged across three seeds.

an extremely high or low mutation rate. Instead, a moderate mutation rate is expected to be beneficial. Too low a mutation rate might not introduce sufficient diversity, leading to early convergence, while too high a rate could make the algorithm behave like a random search. [6]

We tested population sizes ranging from 25 to 1000 and mutation rates from 0.01 to 0.5 to observe changes in behavior. Our results align with expectations: all problem sizes performed best with the highest offered population size, indicating the need for extensive exploration regardless of problem size. From this data, it is likely that when offered a larger population size, the problem will choose it at the cost of convergence speed and computational costs. Fig. 4's average fitness and FEval graph suggest that additional iterations would scale with FEvals, convergence speed and fitness proportionally as the search space grows, which speaks to the robustness of GA at finding the optimal search space.

However, as problem size increases, the mutation rate decreases. This likely occurs because larger problem sizes increase search space complexity. As shown in Table II, a moderate mutation rate for problem size 25 and a lower mutation rate for larger sizes help refine and exploit existing solutions. Excessive exploration in larger problems could hinder the algorithm's ability to converge to an optimal solution, as it may continue exploring new, potentially suboptimal solutions.

Problem Size	Population	Mutation Rate
25	1000	0.2
50	1000	0.07
100	1000	0.05

TABLE II
GENETIC ALGORITHM HYPERPARAMETERS FOR FOUR PEAKS

C. Randomized Hill Climbing

We experiment with a range of 0-64 restarts. We hypothesize that RHC would prefer the upper bound of our restart options, as restarts help RHC avoid getting trapped in local optima by allowing it to begin searching again from different starting points, increasing the likelihood of finding better solutions. Our hypothesis was correct, as all problem sizes performed best with 64 restarts, the highest offered. However, this comes at the cost of computational resources, as a larger number of restarts required more iterations to converge.

As seen in Fig. 5, RHC demonstrates limited effectiveness and robustness for the Four Peaks problem. In particular, smaller problem sizes have higher variance, while larger problem sizes show slightly better fitness and reduced variance, but still relatively low scores. This suggests that RHC's exploration-exploitation trade-off tends to quickly exploit local optima without sufficiently exploring other options. At larger problem sizes, the search space becomes more complex, which forces RHC to perform more exploratory steps before converging at the cost of longer runtimes, but this still only modestly reduces variance and improves fitness.

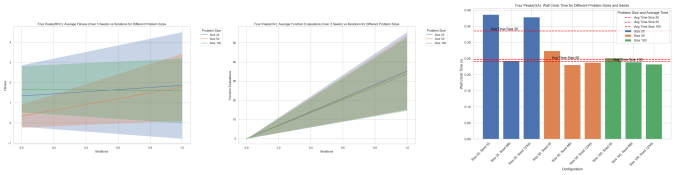

Fig. 5. RHC struggles with the Four Peaks problem, as seen through its high variance.

III. MAX K COLORS

We hypothesize that GA will achieve better fitness values at the expense of higher computational costs, as opposed to RHC and SA. We believe that this comes from the need to evaluate a population of solutions in each iteration rather than the single-solution approaches of the other algorithms. Furthermore, we anticipate that GA will show faster convergence, which can be attributed to its ability to combine and mutate solutions. This allows it to explore the search space more effectively and escape local optima quicker than RHC, which tends to get trapped in local optima, and SA, which which better atescap-

ing local optima, converges slower due to its probabilistic acceptance of worse solutions.

Our methodology here closely follow that which was used in the Four Peaks section above. Likewise, the tuned hyperparameters remain the same.

A. Simulated Annealing

We anticipated that geometric decay would be preferable to exponential and algorithmic decay for larger problem sizes due to its gradual reduction in temperature. By slowly reducing the temperature, the search space can be explored for longer periods before transitioning to exploitation. We expected the temperature to be consistently high for all problem sizes, as this would promote more diverse exploration.

Our hypothesis about temperature proved mostly true, with both large and small problem sizes preferring high temperatures (Table III). For moderate problem sizes, a temperature of 250 was preferred, which, although lower, still aligns with our hypothesis as it falls in the upper range of our temperature spectrum (0.1 to 1000).

Interestingly, geometric and exponential decay produced near-identical results across problem sizes. They both yielded similar fitness scores and displayed similar trends of linear fitness improvement over iterations. Geometric decay showed faster convergence with fewer fluctuations, while exponential decay achieved faster wall clock times. These differences might be attributed to computational efficiency or the limited range of problem sizes. We believe that with more complex problems and larger search spaces, the impact of decay types will become more pronounced.

Problem Size	Temperature	Decay Type
25	1000	Exponential
50	250	Exponential
100	750	Exponential

TABLE III
SIMULATED ANNEALING HYPERPARAMETERS FOR MAX K COLORS

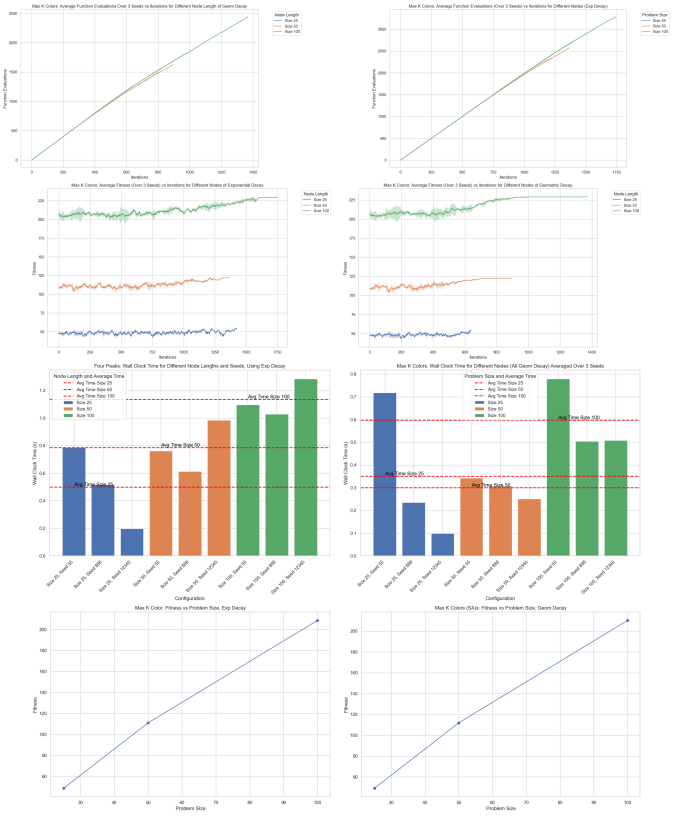

Fig. 6. Side-by-side comparison of exponential (left) and geometric decay on FEvals, fitness/iteration, wall clock times and fitness over problem size for SA.

B. Genetic Algorithm

We anticipate that a higher population size within the provided range for the Max K Colors problem will result in a broader exploration of the search space, while a moderate mutation rate will enhance the solutions by improving the exploration-exploitation trade-off. This combination should help GA avoid local optima by continually introducing new variations to the population.

A common observation is that regardless of the population size (ranging from 25 to 1000), GA tends to converge around 50 iterations and plateau quickly, even though it is allotted 500 iterations. This indicates that GA efficiently explores the search space and identifies optimal solutions early and robustly, with minimal fluctuations post-convergence. This efficient convergence can be attributed to GA's mechanism of balancing exploration and exploitation, allowing it to quickly identify promising areas in the search space.

Our hypothesis about GA's preference for a higher population size proved correct, as it preferred the upper bound of population size over the problem sizes. In the context of Max K Colors, a higher population implies more genetic diversity, which results in a more comprehensive sampling of solutions of near-optimal colorings, even in complex graphs with many nodes and edges.

Our cross-validation search on mutation rates, ranging from 0.01 to 0.5, reveals varying results for different problem sizes. For a node size of 25, fitness increases linearly with higher mutation rates, suggesting that smaller problem sizes benefit from higher mutation rates to introduce more variance and escape local optima. Conversely, for a node size of 100, while higher mutation rates initially benefit larger problem sizes by exploring a broader search space, rates closer to 0.5 resemble random search, resulting in longer convergence times. However, these higher mutation rates still return higher fitness scores, indicating that larger, more complex problems may benefit from more exploration to find optimal solutions.

Our initial hypothesis about GA's performance proves true. GA's performance consistently surpasses SA and RHC in terms of function evaluations (FEvals) and fitness over iterations. For instance, GA maintains the highest initial fitness over iterations, as seen in Fig. 7. However, achieving this requires substantially more FEvals than SA and RHC (Fig. 8), leading to higher computational costs and longer runtimes (Fig. 9). This higher computational cost is expected due to GA's approach of evaluating multiple individuals per

generation. Nevertheless, GA demonstrates robustness by outperforming SA and RHC with lower variance and shorter convergence times, justifying the computational expense.

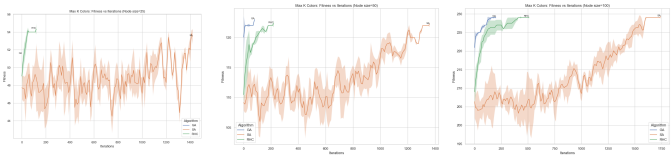

Fig. 7. Averaged fitness/iteration over multiple algorithms for Max K Colors.

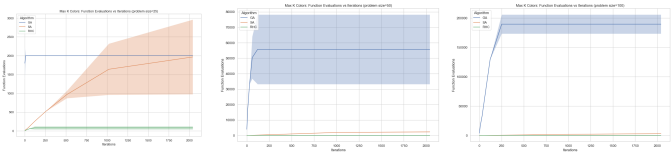

Fig. 8. Averaged FEval/Iteration over multiple algorithms for Max K Colors.

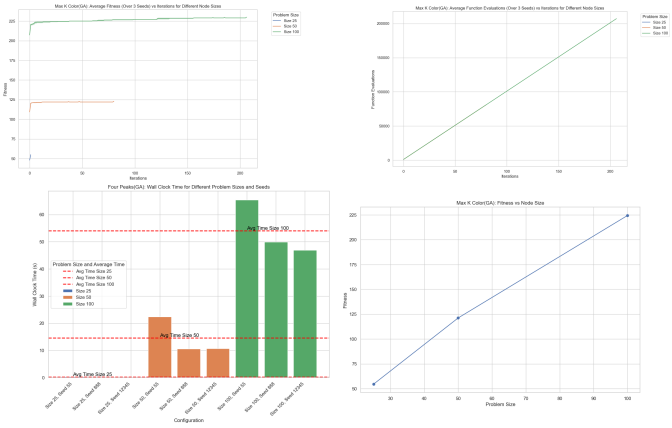

Fig. 9. Plots of fitness/iteration, fitness/problem size, FEvals/iterations and wall clock time for GA problem sizes 25, 50 and 100 averaged across three seeds, for Max K Colors.

Problem Size	Population	Mutation Rate
25	1000	0.01
50	1000	0.4
100	1000	0.5

TABLE IV
GENETIC ALGORITHM HYPERPARAMETERS FOR MAX K COLORS

C. Randomized Hill Climbing

Like with Four Peaks, we believe that the number of restarts needed for the Max K Colors Problem will increase with problem size due to the expanding complexity of the problem space. Since restarts allow RHC to begin the search process from multiple starting points, better performance should be achieved as the likelihood of escaping local optima increases with more restarts.

Interestingly, our cross-validation results suggested that the initial fitness value for 0 restarts is higher than the configuration with restarts. This implies that with no restarts, the initial solution might be relatively good or the algorithm may quickly find a good solution initially. However, as the iterations increase, configurations with more restarts tend to achieve higher fitness values, indicating that the overall solution quality and robustness improve significantly with more restarts, which is in line with our hypothesis.

RHC's average performance across three seeds suggest that it does not do well with problem complexity, as seen by how it requires longer computational effort with increased problem size (Fig. 10). Additionally, as node size increases, so does the required FEvals and convergence time, which suggest that larger problems need more iterations to converge.

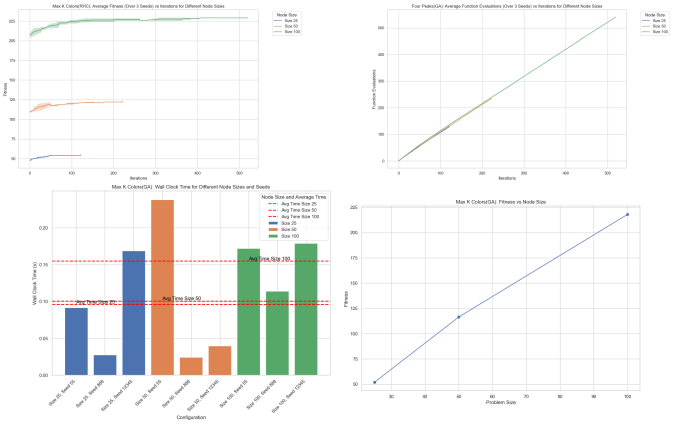

Fig. 10. RHC's performance across problem size suggests it struggles with converging with more complex problems.

IV. WEIGHING A NEURAL NETWORK

For the Abalone dataset, we began with removing rows with missing values and conducting data visualization and exploratory data analysis. Then, we performed one-hot encoding on the 'sex' feature and removed instances where the abalone's age was labeled as "infant," as abalone not mature to receive a sex likely did not have 11 rings. This reduced our dataset to 2834 rows. We also one-hot-encoded our target variable for our binary classification task so that abalones with 11 rings or more had values of 1 or 0 otherwise. Feature scaling through standardization was applied, as well as a subsequent data split into a training and testing set.

In order to do a comparison of backpropagation to the RO's, we use the same parameters for each algorithm (e.g, iterations, seed). We utilize the PyPerch library and stratifiedKFolds cross-validation (chosen because of the imbalance in our target feature) to determine activation function and layer architecture. We reason that the activation function and layer size needed to be recalculated because previous exploration did not use the Pyperch library, which would cause some differences in how the optimal activation function and layer architecture were calculated.

To optimize our weights for each algorithm, we first determine the best layer size and activation function through stratified K-fold cross validation with backpropagation. We freeze the best-found (determined as having returned the highest fitness, with wall clock time taken into consideration)

layer size and activation function for the following algorithms. Afterwards, we optimize hyperparameters unique to their algorithm, such as temperature and cooling for SA, population size and mutation rate for GA, restarts for RHC and learning rate for gradient descent (GD). We plot the learning curves at each step to gain insight into how well the model learns over time.

After each algorithm has been optimized, we compare convergence and F1-score performance. We also adjusted the fitness function so that these algorithms solve a maximization problem instead.

A. Backpropagation with Gradient Descent

We begin by experimenting with different activation functions with 5-fold cross validation. Our thoughts are that ReLU would converge quickly because unlike sigmoid and tanh, reLU does not suffer from vanishing gradients which allow the weight to updated more effectively during backpropagation [7].

From Fig. 11, we see that our training loss and validation loss curves demonstrate that reLu activation results in a rapid loss in the initial training phase and flattens around 100 iterations, indicating quick convergence and a steady state. Additionally, the low and steadily decreasing training loss indicates that the model is learning effectively and not underfitting, implying low bias. The gap between the training and validation curves for both loss and F1 score is small, which indicates that the model generalizes well to unseen data. It seems that our hypothesis about reLu was proven correct.

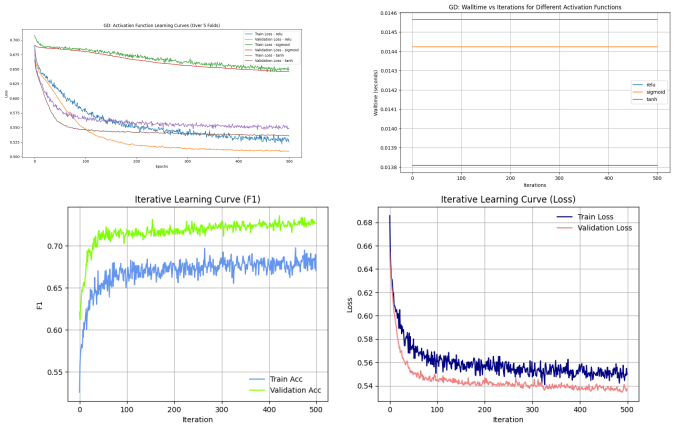

Fig. 11. Learning, training and validation loss curves and wall clock time for different activation functions.

We repeat a similar process to analyze layer configuration. We do not expect that the abalone dataset will require a complex configuration, considering the limited amount of features and a relatively small dataset, which should allow for the model to learn from te data without overfitting.

Our analysis support our thoughts. By analyzing the loss, it seems that among complex architecture of multiple layers with weights up to 256, single and double layer architectures of (64) and (64, 64) respectively were preferred in terms of convergence and loss values. Though they converge similarly, we move forward with (64) since it requires less wall clock time.

We analyze learning rates since they determine the speed and quality of the training process and adjust the weights with respect to the gradient of the loss function. We hypothesize that a high learning rate will do poorly, since the training process may get stuck in a local minima or converge quickly. From Fig.12, we see that higher learning rates converge faster but exhibit higher fluctuations, which suggest that variance is higher and may be potentially overshooting the minima.

By comparing the loss curves of Fig. 11, with just our initial hyperparameter, the activation function, we see improvement in Fig. 12's loss curve, as the training and validation lines move closer.

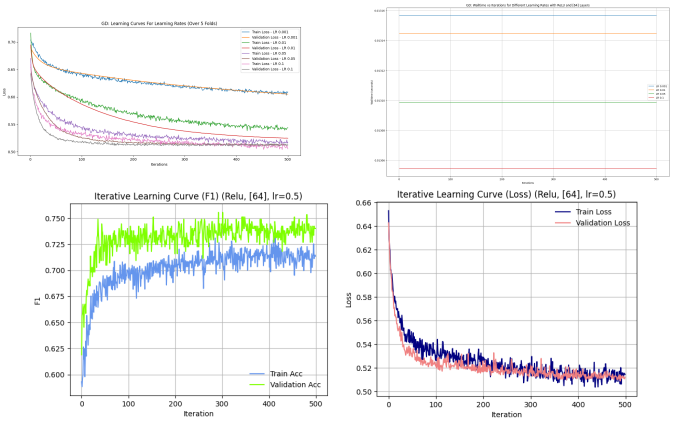

Fig. 12. GD learning rate learning and loss curves show that the optimization process has improved the weights.

B. Simulated Annealing

For simulated annealing, we begin by gathering a baseline for performance using the parameters set by GD: reLu activation function with a single-layer architecture.

While less smooth than backpropagation's curve in Fig. 11, SA's initial learning curves (Fig. 13) indicate that there is steady convergence at around 500 iterations. We suspect that the convergence takes longer to backpropagation because unlike GD, SA prioritizes exploration and accepts worse solutions by using temperature to determine its probability, which leads to slower convergence. In comparison, GD is exploitation heavy, as it always tries to minimize the loss function.

Fig. 13 also illustrates that the model generalizes well to new data, as the gap between the training and validation curves are minimal and demonstrate low variance.

Next, we hypothesize that a larger temperature would increase the iterations but overall improve convergence by allowing the algorithm to transition from exploration to exploitation and have time to thoroughly search the solution space. Furthermore, we believe that a higher cooling schedule would allow for more exploration, which would be beneficial in this case as the problem is complex.

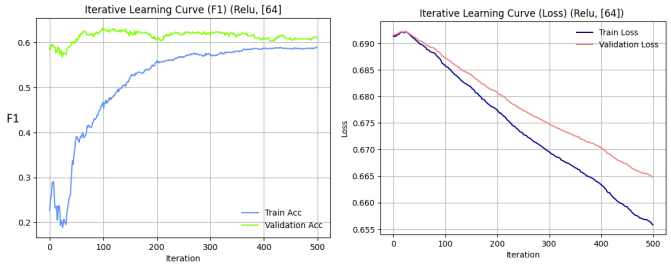

Fig. 13. Initial learning curve of just base activation function and architecture for SA.

We test our hypothesis with cross-validation and discover that adjusting the temperature did not optimize the weights, since there is consistent increase in loss and fluctuating F1 scores, which suggest SA is struggling with high variance and overfitting instead.

This pattern continues into Fig. 14, after which we have determined a cooling rate based on the wall clock time and convergence. Our learning curves suggest that our model underfits our data and the large fluctuations in our training and validation lines indicate that it sees high variance instead. Furthermore, convergence is no longer stable, as seen by the high variability in F1 scores. We beleive that this is due to the chosen temperature and cooling settings. Specifically, that the chosen temperature was too high, which led to excessive exploration without sufficient exploitation and does not allow the model to stabilize.

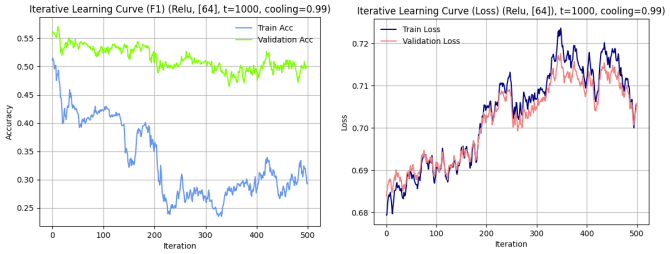

Fig. 14. Our cooling and temperature adjustments for weights calculated with SA take us away from optimization and rather overfit the model instead.

C. Genetic Algorithm

For our GA portion only, we reduce the search size down to 3 folds and 50 epochs (as opposed to the original 500). However, because this specific algorithm was adjusted to use lower epochs, we recalculate optimal activation function and layer size. While the layer size remains the same, surprisingly, the optimal activation function changes to sigmoid instead. We anticipate that this is because sigmoid provides a smooth output gradient, which could help explore the space more effectively than reLu, which could have abrupt changes.

To reduce computational resources, we limit our population size to a range of 50 to 200. We anticipated that a population size of 200 would help to reduce the risk of getting trapped in local optima, while also allowing for better converging by focusing on exploitation. However, cross-validation results returned that a population of 100 is a better exploration-exploitation trade-off. Since the difference between a population size of 100 and 200 is not that stark, we can assume that it is due to noise in the data.

We also adjust mutation rate, tested from values of 0.01, 0.05 and 0.1. We anticipated that a high mutation rate would not be optimal since it would be similar to random search, and not useful in the task of binary classification. However, the iterative learning curves generated from the combination of parameters, that although they show variability in scores, are consistently high (Fig. 15). This, coupled with the fact that the loss curves for different mutation rates have low values and minimal fluctuations, suggest that mutation rate does not drastically impact SA convergence. Rather, it has little impacted, which is supported by the fact that the optimal mutation rate is 0.01, which supports our hypothesis.

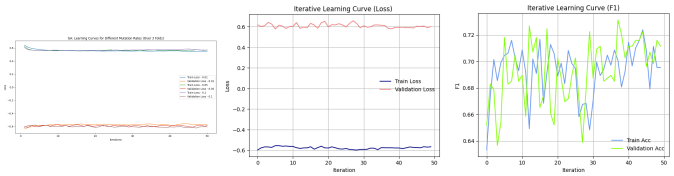

Fig. 15. Training, validation and loss curves suggest that mutation rate has little impact on the dataset.

D. Randomized Hill Climbing

For RHC, we return to using 5-fold cross validation, 500 iterations and a reLu activation function with a single-layer architecture. Here, we only optimize restarts, since we hypothesize that the Abalone dataset does not actually need restarts considering the data size.

In Fig. 16, we see that is indeed the case. The loss curves for different restart configurations all converge, indicated by their stable fluctuations. However, the high training F1 score with the overlapping validation F1 curve, suggest that there is high bias and that the model is not complex enough to capture the underlying pattern in the data. This is most likely caused by the dataset itself, which does have correlated features and non-linear relationships between them.

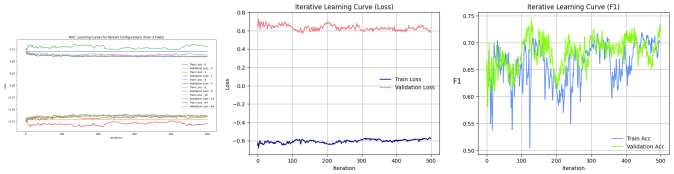

Fig. 16. RHC does a poor job at optimizing weights, as indicated by its high bias.

E. Algorithm Comparison on Test Set

Fig. 17 shows that backpropagation has the best performing learning curves in that it showcases the highest fitness along with the lowest variance, denoted as distance between the training and validation scores. While SA and GA could be considered contenders, they fall short in terms of fitness

curve and convergence. In comparison, RHC appears even less effective due to its inability to smoothly converge.

We hypothesize that backpropagation likely performs the best on the Abalone dataset due to its ability to fine-tune the weights more precisely through gradient descent. Additionally, since the Abalone dataset is not an inherently complex dataset, the exploratory capability of algorithms such as SA, GA and RHC could be excessive. In comparison, backpropagation's straighforward ånd deterministic optimization approach appears to handle these simpler patterns more effectively.

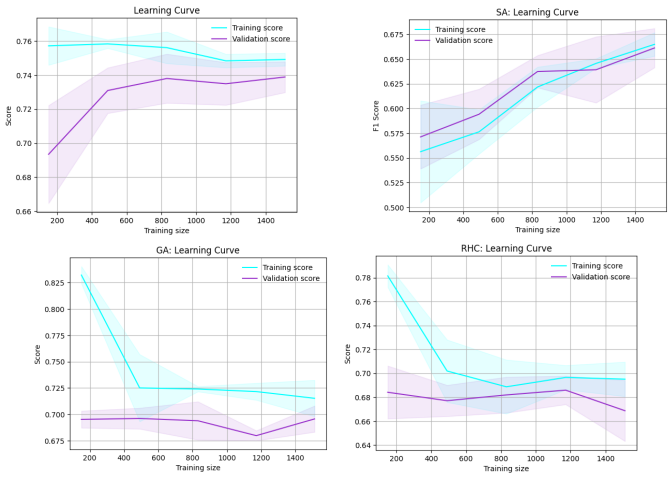

Fig. 17. Comparing algorithm weights for NN on Abalone dataset.

V. Discussion

To enrich this report and to ensure robust results, several adjustments could have been made. For instance, for the Four Peaks and Max K-Colors problem, a pipeline could have been used to prevent any data loss. Future work could be done to ensure that the solutions generated by the algorithms are robust and more in-depth analysis on convergence and relation to hyperparameters could be given. Additionally, when determining optimal weights for the NN and finding that the returned value from cross-validation did not optimize our weights, more granular and manual tuning could have been done.

VI. Conclusion

The extensive analysis of using random optimization algorithms (RO) on discrete optimization problems, such as Four Peaks and Max K-Colors, revealed key insights into the impact of randomization and hyperparameter tuning. We implemented three RO algorithms—Randomized Hill Climbing (RHC), Simulated Annealing (SA), and Genetic Algorithm (GA)—on maximized versions of these problems. Using the mlrose-hiive and Pyperch libraries, we fine-tuned the algorithms and generated hypotheses for each algorithm's hyperparameters.

Our findings show that SA excels in exploring the search space with fewer function evaluations in Four Peaks, while GA performs best in Max K-Colors, achieving higher fitness scores, faster computational times, and fewer function evaluations compared to SA and RHC. Additionally, we explored the impact of RO algorithms on optimizing neural network (NN) weights for the Abalone dataset. Backpropagation demonstrated superior performance, likely due to its straightforward minimization of the loss function.

Overall, our experiments highlight the importance of randomization, convergence, and the factors that influence them in optimization tasks.

References

[1] Qian, C., Bian, C., Jiang, W. et al. Running Time Analysis of the ( )-EA for OneMax and LeadingOnes Under Bit-Wise Noise. Algorithmica 81, 749–795 (2019). https://doi.org/10.1007/s00453-018-0488-4 *Journal of Machine Learning Research* 12 (2011): 2825–2830.

[2] Nash,Warwick, Sellers,Tracy, Talbot,Simon, Cawthorn,Andrew, and Ford,Wes. (1995). Abalone. UCI Machine Learning Repository. https://doi.org/10.24432/C55C7W.

[3] https://github.com/jlm429/pyperch/tree/master

[4] C. Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P. (1983). Optimization by simulated annealing. Science, 220(4598), 671-680. doi:10.1126/science.220.4598.671

[5] https://github.com/hiive/mlrose/tree/master

[6] Goldberg, D. E. (1989). Genetic algorithms in search, optimization, and machine learning. Addison-Wesley.

[7] V., Nair, G. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines."*Proceedings of the 27th International Conference on Machine Learning* (2010): 807-814.