Assignment 2 Report

Triveni Bhat
Department of Computer Science
Georgia Institute of Technology
Atlanta, USA
trivenibhat@gatech.edu

I. Introduction

I have chosen Four peaks and Flip Flops as two problems on which I run the random optimization problems. The goal of the problem is to find the best binary string find the maximum values of the fitness function. The functions considers the number of 1s in the initial part of the string and trailing 0s for the problem and a peak function with a threshold. Tries to find the nice balance between the two, to find the best string the yields the best value for the fitness function. The other problem I have experimented is Flip Flops. A Flip flop occurs when 2 consecutive bit in the string differ from each other. For example when one bit is 1 and the next one is 0 "101010101". The goal of this optimization problem is to find the binary string that maximizes the number of these flip flops. These problems are important to experiment with because four peaks provides a challenging search space where there should be a balance between exploitation and exploration. Flip flop is a simpler problem but it has a lot of local optima, serving as a standard benchmark problem that could be used to tune the hyper-parameters of the algorithms to get the global optimum. Both provides different aspects that needs to be tuned for the best performance of the optimization algorithm.

II. Hypothesis

I am hypothesize that the algorithms will find the best fitness values or global optimum for both of the problems. In terms on Four peaks, I believe that SA will perform the best because the problem focuses on the trade off between exploration and exploitation. I believe SA is the best fit for such types of problems as the exploration takes place in higher temperatures of the algorithms and as it cools down the algorithm exploits the current peak for the best value. GA would perform overwhelmingly because the population could take a lot of generations to reach the desired level of fitness, because the search space is quite large. For Flip Flops I believe that GA will perform better with a lot of candidate solutions. This is because the algorithm needs to escape for a lot of local optima, having many candidate solutions can help explore the most of the local optima in order to find the global optimum. In terms of time, I think GA will take a lot of time to converge. Because it starts of with a certain population size and more number of function evaluations are needed for GA. Whereas SA has only one bit string as a candidate solution and takes less number of calculations and hence less time to converge. As the size grows it becomes extremely hard to hypothesize which algorithm would perform better, however I would suggest that GA would do marginally better because of it population size and hence is more probable to perform better as the size increases. SA however with a bit string would take quite some time to find the best answers as the problem size increases.

III. Experiment Methodology

I have fixed the maximum iterations to be 1000 and max attempts to be 100 with the intention of being fair for all algorithms to be able to directly compare how their converge under the same conditions. Additionally during hyper-parameter testing and generating graphs relating to iterations, the problem size is set to 20 for both of the problems. The convergence behavior changes for the iteration constraints and whether some algorithms are able to find optimal solutions faster or need more iterations. Additionally, you can analyze the impact of this cap on performance metrics such as fitness and runtime across your experiments.

IV. Four peaks

A. GA

For Genetic Algorithm, I first tried to find the best mutation probability. I had different seeds(numbers) to vary the random states. I took the mean of the best fitness value across different seeds, for each mutation probability. I followed the same procedure to find the best population size. The best mutation probability was 0.2 and the best population size was 10 however I noticed that the fitness score was stagnant. This gives the impression that the Fitness values are independent of the mutation probability. But, the behaviour could be because GA is not able to explore all the Fitness values. To test this I increased the max iterations to 2000 and re-ran the same experiment. The best population size was 50 and the best mutation probability was 0.10 the fitness score was 36.0. The mutation probability was again stagnant and it seems for a population size of 50 the mutation probability is independent of the fitness score.
From this experiment I conclude GA takes a lot of iterations to convergence. GA has a lot of candidate solutions and hence would take more time to explore the solution space. Population 20 was the best because: The four peaks problem has a few regions in the search space that yield very high fitness scores. If we look at the population size of 20, it seems to be sufficient to inspect the search space. The subsequent population sizes

are all more than 20 but they do not provide any improvement in the fitness. One reason is as the population increases there is a lack of diversity which can cause the algorithm to not explore the search space efficiently and settle on a local optimum.
The mutation probability gives a plateau of fitness scores on a population size of 20. The size might not be large enough to maintain sufficient genetic diversity across generations. With not enough diversity the algorithm may converge to a sub-optimal solution. No matter what mutation probability is chose for the small population the fitness value seems to be stagnant. In the context of the problem, the algorithm must search efficiently as the search space is 2**20 which is quite a big number.

B. Other Algorithms

I experimented with the schedules of the Simulated Annealing. Geometric and Exponential schedule performed the best. Geometric cooling lowers the temperate faster than Arithmetic, Exponential scheduling reduces the temperature faster than Geometric. It is essential for this problem to decrease the temperate faster to do more "exploitation" than exploration. The initial state was fixed an it tried experimenting with different restart values: 10, 20 , 50, 100, 200, 500 , 1000 , 2000, 5000. The fitness value reaches above 35 for restart 200 when the number of restarts is at 1000 and then stays stagnant. Once RHC reaches a peak, additional restarts are less likely to find something much better unless they start near another peak. Even with restarts, if most starting points land in regions near local optima, the algorithm can repeatedly get stuck at sub-optimal solutions like a fitness of 36. Hence, RHC is prone to getting stuck in local optima. For MIMIC I explored the different hyper parameter settings using grid search.

C. COMPARISON

I took 10 random seeds and the best hyper parameters obtained from the previous section. I calculated the mean and standard deviation of the best fitness values for each random seed.

1) Mean fitness vs Iterations: In the figure 1, RHC can be seen to only run for about 50 iterations, RHC is likely to get stuck in a local optima. Mimic runs for 100 iterations and initially it performs better and degrades over time. GA converges later than RHC and MIMIC but provides better fitness score. SA takes a lot of iterations to converge, and it fails to give a better score than Genetic Algorithm. This happened because Genetic Algorithm has 50 candidate solutions and can explore better than RHC and SA where the focus is on one candidate solution. The mutation rates and crossover blends candidates with high fitness score to increase the chance of finding better solutions. GA performs better than MIMIC because GA introduces diversity in the population enabling it to search in different spaces. MIMIC starts to narrow its search space very quickly because it is based on a small subset of top solutions. It it likely that it reaches for a sub-optimal solution.

2) Mean feval vs iterations: In Figure 2, as the iterations increases Function evaluations for GA increases exponentially because of a large population size. This causes a lot of computational power on GA. RHC has a downward trend for function evals because of its quick convergence. SA has a steady growth but does not converge in 1000 iterations and mostly looks stagnant. Mimic has the largest number of feval and it scales to a much higher values as the number of iterations increases. In terms of fevals, SA and RHC performed the best with the least number of fevals. MIMIC has the costliest fevals but GA does comparatively better than MIMIC.

3) Fitness values vs size: The focus is on the sizes 3-50 of the bit strings. Initially all the algorithms perform equally well and they start to diverge from the size 10. It is quite hard to determine which algorithm performs the best because there is a lot of variance. It can be noted RHC has a downtrend as the size increases. Mimic has a uptrend. SA performs marginally better than GA albeit they both fluctuate a lot. The reasons for GA fluctuation could be that it is hard for the algorithm to converge within 1000 iterations. As the size grows the iterations it needs to converge should also increase. Because the search space increases exponentially as the size increases. Hence it is highly likely that GA is setting stuck in sub-optimal peaks.

4) Fevals vs size: One more aspect is to see how long the algorithm takes to run as the problem size increases. From Figure 4, we can see that MIMIC wall clock time increases as the problem size increases. However SA, GA , RHC seem to remain constant as the problem size increases. GA can be see to slowly rise up from the other two algorithms. GA's time increases modestly as the problem size increases because of the population size. As longer bit string take more time to mutate and cross over GA needs more time as the size grows.

D. Verdict

In my initial hypothesis, I believed Simulated Annealing (SA) would perform best for the Four Peaks problem due to its ability to balance exploration and exploitation. I anticipated that as SA's temperature cooled, it would effectively exploit the search space, while Genetic Algorithm (GA) would struggle with large problem sizes due to requiring many generations to converge. However, my experiments showed that GA actually performed better for smaller problem sizes, providing higher fitness values, albeit at the cost of longer computation time. As the problem size increased, SA did indeed perform marginally better in terms of time efficiency, but GA's population-based approach consistently delivered stronger results.

This outcome demonstrates the trade-off between exploration and convergence time. While GA starts with multiple candidate solutions, which helps it converge earlier, it also requires more function evaluations, making it slower. SA, on the other hand, benefits from faster convergence with fewer computations, but its single-solution exploration makes it less effective in finding optimal solutions for smaller problem sizes. This experiment revealed that GA is more suitable for small to moderate-sized problems, while SA becomes more practical

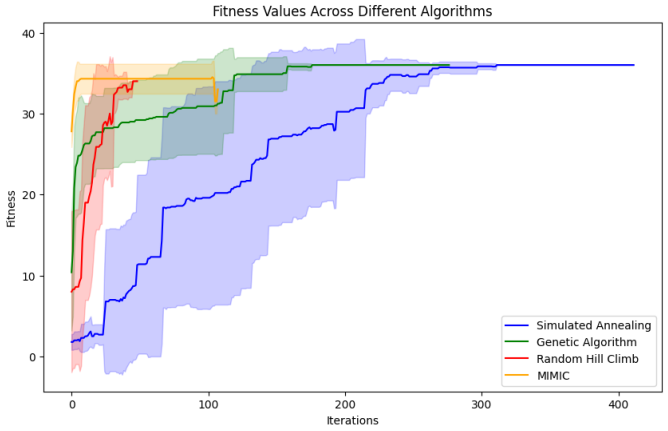

Fig. 1. Mean Best Fitness value vs Iterations

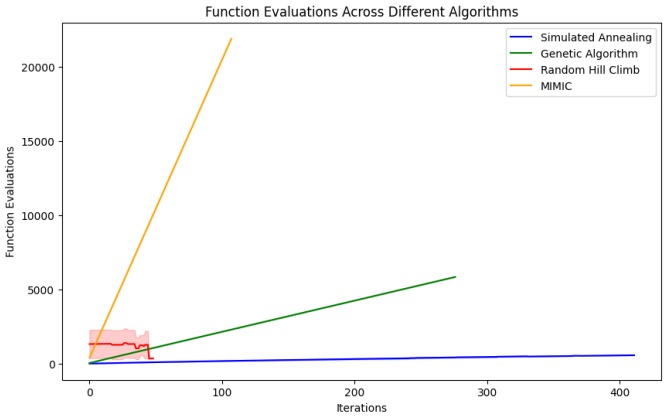

Fig. 2. Mean best function evaluations vs Iterations

for larger sizes, disproving my initial assumption that SA would consistently outperform GA.

V. FlipFlop

A. SA

I experimented with different schedules of SA. The Geometric and exponential decay schedule outperform Arithmetic schedule due to the way they handle solution space exploration. Geometric and Exponential decay schedule allow the temperatures to be decreased in a quick rate which helps the algorithm to get out of bad solutions in the beginning. However due to a longer time in lower temperatures, the algorithm has along time to refine its solution. This is also helpful as it prevent from premature convergence. For the problem, it is imperative to drop the temperatures slowly because that means that the algorithm stays in high temperatures for a long duration. The search space has a lot of local optima. So, it is essential to maintain high temperatures for a longer duration where the algorithm can accept random transitions to explore the search space efficiently. Arithmetic scheduling performs better than the other cooling schedules.

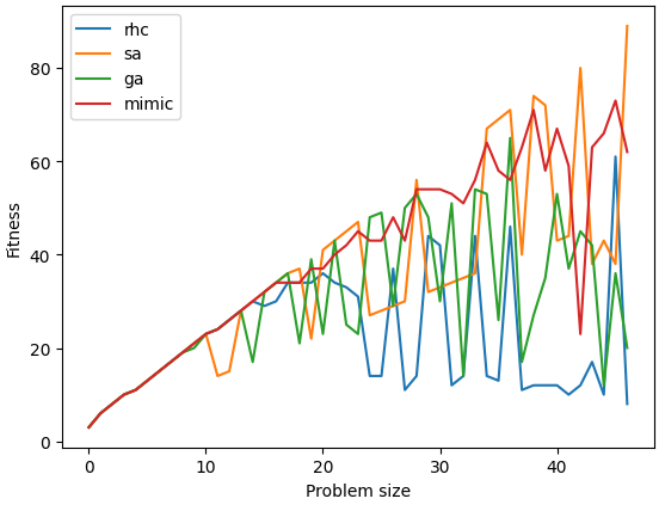

Fig. 3. Best fitness vs size

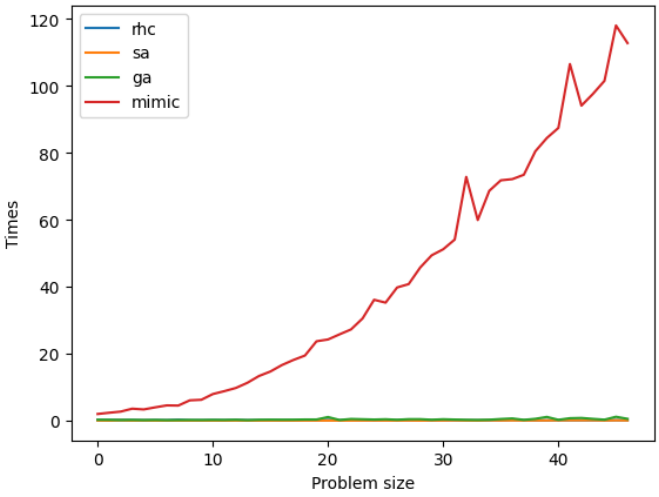

Fig. 4. Time vs size

B. Other Algorithms

For GA, tuning the parameters like the population size and the mutation probability is quite important for the algorithm. Hence I focus on tuning the two of them. Best mutation probability was 0.1 and the best population size was 20. Greater mutation probability did not reach the global optimum. This is because as the mutation probability increases it is possible that the strong solutions were forced to change itself or mutate, this resulted in losing strong candidate solutions and the algorithm did not reach the global optimum. I had also used Random Hill climbing and tried to tune for the best restart hyper parameter. I tried 10, 20 , 50, 100, 200, 500 , 1000 , 2000, 5000 values. The best restart value was 1000 for the fitness value of 19 which is the global optimum. For mimic I used grid serach to find the best hyper parameters.

C. comparison

1) Fitness values vs iterations: In the Flip Flop problem, the graph shows that RHC (Randomized Hill Climbing) and MIMIC quickly attain the global optimum, both converging within 50 to 100 iterations. Their efficiency in reaching the global solution early is due to their focused search strategies: RHC performs greedy local improvements, while MIMIC models dependencies between variables, allowing it to sample high-quality solutions efficiently. These methods rapidly explore the solution space and converge faster than the other algorithms.

In contrast, GA converges around 100 iterations but only reaches a local optimum, rather than the global maximum achieved by RHC and MIMIC. Despite its population-based approach providing diversity and helping escape local optima in many cases, GA falls short of reaching the global best here, likely because it prematurely converges before exploring enough of the search space. After about 100 iterations, GA settles at a sub-optimal solution.

Simulated Annealing (SA), however, takes far longer to converge and doesn't reach a stable solution even after 1000 iterations. It ultimately settles at the same local optimum as GA. SA's slower convergence is due to its probabilistic exploration, where it spends a significant amount of time searching the space at higher temperatures before exploiting solutions. Unlike GA, which converges earlier, SA struggles with longer exploration phases and only settles after extensive iterations, still failing to surpass GA's performance.

2) Fevals vs iterations: Figure 6 illustrates the number of function evaluations vs iterations. RHC and MIMIC show a sharp spike in function evaluations early in the process, indicating their efficiency in finding solutions quickly. Both algorithms converge within the first 50-100 iterations. RHC, being a greedy algorithm, quickly escalates to the local peak, while MIMIC samples from the population in a way that models dependencies, allowing it to find optimal solutions efficiently. Despite using many evaluations early, their quick convergence indicates they can find optimal solutions with minimal iterations.

In contrast, GA shows a more gradual increase in function evaluations, with steady progress until it levels off around 100 iterations. GA evaluates multiple candidates at each iteration, which results in a higher number of function evaluations compared to SA but fewer compared to RHC and MIMIC. It converges more slowly but can still explore diverse areas of the search space. On the other hand, SA displays the slowest growth in function evaluations, progressing linearly throughout 1000 iterations. This slower pace is due to its probabilistic nature, where it explores more gradually, accepting worse solutions at the beginning as part of its cooling schedule. Consequently, SA takes much longer to converge, explaining why its function evaluations remain lower than the others, but it also means it doesn't reach optimal solutions as quickly as RHC or MIMIC.

3) Fitness vs size: The fitness values for each algorithm remain constant across all problem sizes in Figure 7. This is common in problems like Flip Flop, where once an algorithm identifies a solution structure that maximizes fitness, then for the increasing sizes it can just add on to the previous solutions. Therefore, the fitness values stabilize across different problem sizes.

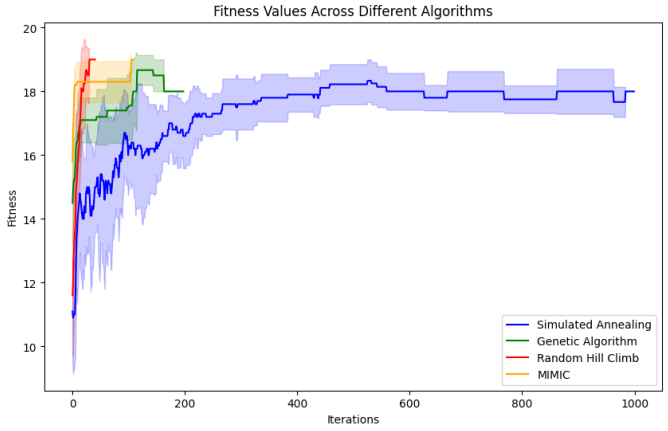

Fig. 5. Mean Best Fitness value vs Iterations

For SA, represented by the orange line, the fitness value remains at 19.0, consistently performing better than both RHC and GA regardless of the problem size. This indicates that SA is capable of maintaining high fitness across various problem sizes. This goes to show that GA can perform well as the size increases. MIMIC performs pretty well by keeping up at 18. RHC, represented by the blue line, performs similarly to SA at a fitness value of 19.0, indicating that while it performs well.. On the other hand, GA, shown by the green line, consistently under performs the other algorithms, maintaining a fitness of 16.0. This suggests that GA, in this particular experiment, may not scale well with problem size or struggles to escape local optima.

D. Time vs size

MIMIC takes a lot of time for the function evaluations, hence the same is reflected in the Figure 8, where x axis represents problem size and y axis represents the time taken for the algorithm. RHC takes slightly longer than SA and GA. SA performs slightly better than GA in terms of time taken as the size increases. This is because of its cooling schedule. SA evaluates one solution at a time and gradually reduces its exploration, focusing more on exploitation as it cools down. This allows SA to converge more efficiently as the size grows

E. Verdict

For the Flip Flops problem, I believed that Genetic Algorithm (GA) will perform better SA. However, GA is expected to take longer to converge due to its reliance on a population size that requires more function evaluations. In contrast, Simulated Annealing (SA) outperforms GA in both convergence time and optimal fitness values as the problem size increases, since it operates with a single candidate solution, requiring fewer calculations. While GA may excel at exploring local

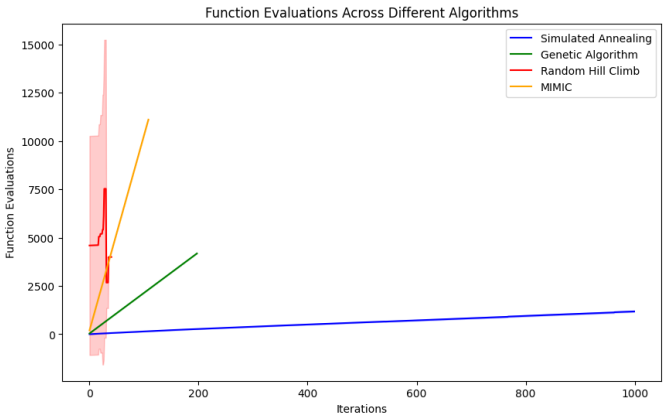

Fig. 6. Mean best function evaluations vs Iterations

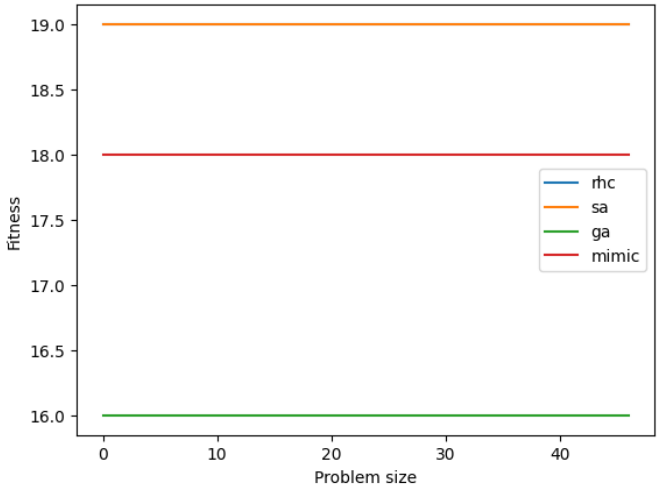

Fig. 7. Best fitness vs size

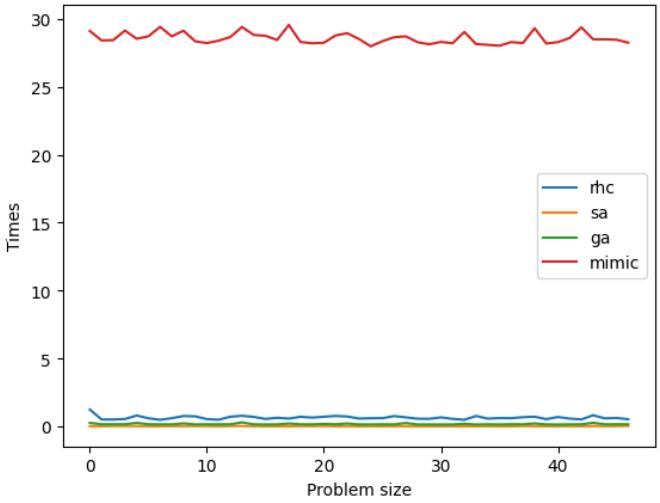

Fig. 8. Time vs size

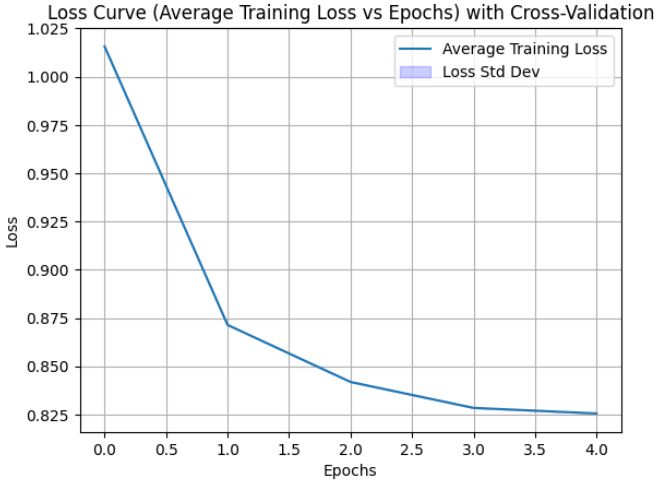

Fig. 9. loss curve for the model freezed in Assignment 1

optima, SA's efficiency makes it the superior choice for larger problems.

VI. ABALONE DATASET

A. Dataset

I have used the Abalone dataset from Assignment 1. I chose this dataset particularly because the performance of my neural network with 150 hidden layers was around 60%. This relatively low accuracy suggests that the dataset is not easily separable, presenting a challenge for traditional optimization methods. To address this, I aim to investigate whether random optimization techniques can enhance the performance of my model. The loss curve for the dataset is in the figure 9

B. Second Hypothesis

Weight optimization methods like Gradient Descent can only be used on a convex function loss functions. Given that the neural network weights are multi dimensional and their complex structure, random optimization can help us find the best set of weights in the complex function space. I believe that SA would perform the best and can help boosting the performance of my current neural network architecture which contains a single hidden layer with 150 neurons. SA starts off at high temperatures exploring the search space. The fitness search space for Abalone could be quite tricky given that the dataset is not easily separable and I expect the search space to be rugged. SA would do especially well in the previous section for a search space with a lot of local optima. The performance measure is the final fitness F1 score on test dataset.

C. Assignment 1 performance

I have identified the best hyper-parameters for my neural network using MLPClassifier, achieving an F1 score of approximately 0.66 with the following configuration: activation=relu, alpha=0.005651157902710025, hidden layer sizes=(150,), solver=adam. To further evaluate the effectiveness of these parameters, I applied the same hyper-parameters

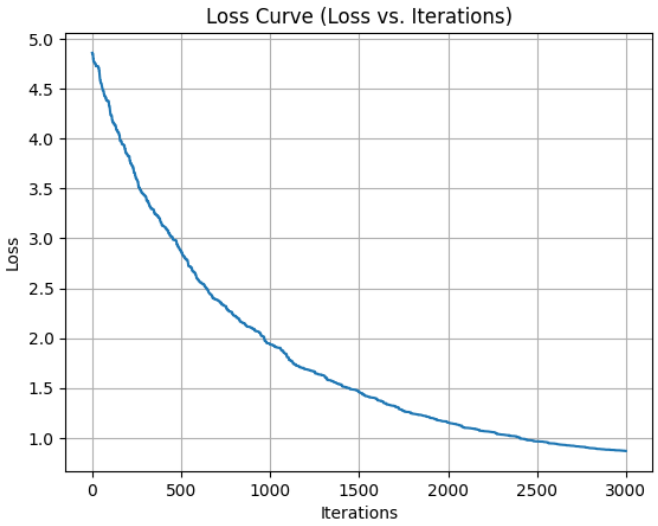

Fig. 10. RHC loss curve

Fig. 11. SA loss curve

in the mlrose.nn framework using the gradient descent algorithm. The model trained with these hyper-parameters yielded an F1 score of approximately 0.13. The drop in the performance is because of the difference in optimization strategy and regularization approach.

D. RHC

I conducted a grid search on Randomized Hill Climbing (RHC), tuning the parameters: learning rate, maximum iterations, and restarts. The optimal configuration identified was restarts = 5, max iterations = 1000, and learning rate = 0.1. With these parameters, the model achieved a validation F1 score of 0.60.

To further enhance the model's performance, I increased the maximum iterations to 3000, which led to an improvement in the validation F1 score to 0.63. The test F1 score, however, stabilized at 0.61. The loss curve shows a consistent decline, from 5 down to 1, resembling the pattern in Figures 9. This suggests that the model continues to reduce loss effectively but may be converging on a local optimum.

MLPClassifier's Adam optimizer uses gradient descent, which combines momentum and adaptive learning rates to more successfully escape local minima.While increasing the number of iterations in RHC improved performance, it also took more time to converge. On the other hand, Adam adapts its learning rate dynamically, which often leads to faster convergence and more efficient navigation of the loss landscape.

To Further improve the performance, we can re-tune the restarts and learning rate by including finer values like 0.001, 0.0001, etc. I think I would also help if the max iters was increased to 5000 or 10000.

E. SA

After tuning, the best performing configuration for Simulated Annealing (SA) was found with the following parameters: max iters=3000, learning rate=0.1, and a GeomDecay schedule. The loss curve initially exhibits an upward spike, reflecting the exploratory phase of SA where worse solutions are accepted to escape local optima. However, as the algorithm progresses and the temperature cools, the loss settles below 0, indicating convergence to a more optimal solution. The final results show a validation accuracy of 0.598 and a test F1 score of 0.588. The initial upward spike in the loss curve noticed during the SA is due to the algorithm's exploration phase. In the early phases, when the temperature is high, SA is more likely to accept sub-optimal solutions in order to explore the solution space and avoid local optima. This conduct produces the initial increase in loss. As the temperature drops and the algorithm gets more discriminating, it increasingly accepts only better answers, resulting in the loss finally falling below zero. The rising trend indicates SA's tendency to prioritize exploration over refining. There are various techniques for improving performance. First, altering the temperature decay schedule or Geometric Decay's decay rate could help balance exploration and exploitation. Increasing the maximum number of iterations beyond 3000 may provide SA with more opportunities to develop its solution. Adjusting the learning rate

F. GA

After conducting a grid search, the Genetic Algorithm (GA) identified optimal hyper parameters: a learning rate of 0.1, a population size of 20, and a maximum of 2000 iterations. To further enhance performance, I increased the maximum iterations to 3000, which yielded improved results. The validation dataset achieved an accuracy of approximately 0.638, while the F1 score on the test data was around 0.628.

The performance of the GA can be attributed to several factors. Firstly, the balance between exploration and exploitation is crucial; a population size of 20 allows for a diverse set of solutions while also enabling effective search through the solution space. Increasing the maximum iterations from 2000

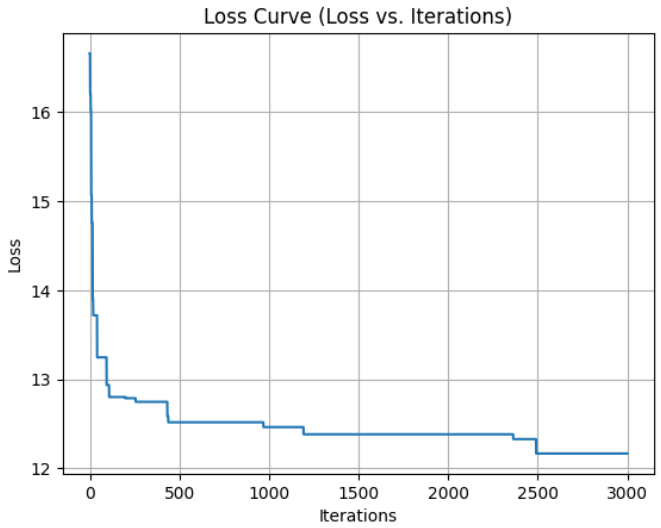

Fig. 12. GA loss curve

to 3000 likely provided the algorithm with more opportunities to explore potential solutions, allowing it to escape local optima and refine its search. The resulting validation accuracy of approximately 0.638 and a test F1 score of around 0.628 indicate that while the model is performing reasonably well, it may still be facing challenges in generalization, possibly due to the dataset's complexity or inherent noise. There is still room for improvement, by adjusting the learning rate and population size.

G. Conclusion

SA did not perform well for the dataset, I believe it was because of the vast search space, SA could not converge to a good solution within the max iterations provided.

GA maintains a population of potential solutions, which help to explore multiple regions of the solution space simultaneously. The diversity introduced through mutation and crossover helps it avoid local optima, leading to better solutions. Additionally, it enables GA to look at new candidates which can result in discovering superior configurations that other algorithms might overlook. GA performed better than the other two optimization methods but was unable to outperform the Adam optimizer due to several inherent differences between these algorithms. Adam on the other hand uses momentum and gradient descent to get to the best solution. This can lead to quick convergence and is faster than GA. The slow steps it takes allows the weights to adjust itself ever so slightly until it reaches a space with no improvements. To summarize for the dataset challenges Abalone provides, the random weight optimization did not perform better than Adam. However with more computational space and hyper parameter tuning we might be able to improve the random optimization algorithm's performances.