

# Assignment 2

Name: Amr Ali  
gtID: 903850635

All code and data present at: <https://github.com/aali343/as2>

## Introduction

In this assignment, we will explore the behavior of several randomized optimization algorithms on several 3 chosen fitness functions.

## Algorithms

We will be investigating the behavior of the following algorithms, each with a set of tweaked hyper parameters.

- Randomized Hill Climbing:
  - No hyperparameters involved
- Simulated Annealing
  - Initial Temperature varied between (0.5, 0.8, 0.99)
- Genetic Algorithms
  - Mutation rate: 0.001, 0.2, 0.4, 0.8
- MIMIC
  - Number of samples kept at each iteration: 0.1, 0.2, 0.5

# Fitness Functions

## Four Peaks

This problem has 2 local optima and 2 global optima, defined through:

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

where:

- $tail(b, x)$  is the number of trailing b's in  $x$ ;
- $head(b, x)$  is the number of leading b's in  $x$ ;
- $R(x, T) = n$ , if  $tail(0, x) > T$  and  $head(1, x) > T$ ; and
- $R(x, T) = 0$ , otherwise.

**Parameters:** `t_pct` (float, default: 0.1) – Threshold parameter (T) for Four Peaks fitness function, expressed as a percentage of the state space dimension,  $n$  (i.e.  $T = t_{pct} \times n$ ).

This problem is interesting in nature as it presents pitfalls that algorithms like RHC and SA can run into.

## FlipFlop

This is a “problem that counts the number of times of bits alternating in a bit string” where “A maximum fitness bit string would be one that consists entirely of alternating digits” (2020).

FlipFlop could potentially give algorithms with learned experience (such as MIMIC) an edge over others

## Count Ones

This function simply aims to maximize the amount of ones in a bit string.

$$Fitness(x) = \sum_{i=0}^{n-1} x_i$$

This problem is very simple in nature, it does not have any local minima; we should look to see RHC and SA shine on this one.

# Results

## Four peaks

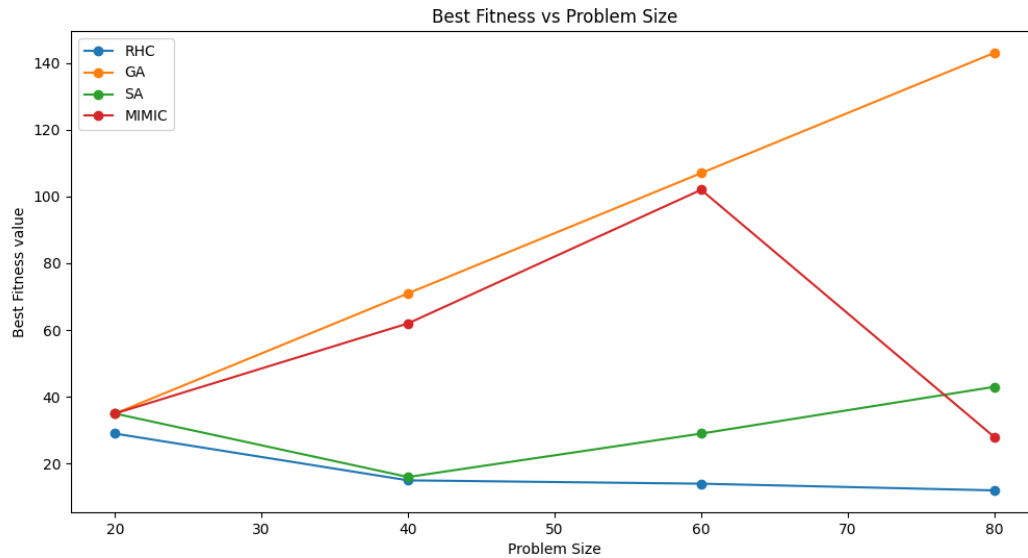


Figure 1. Fitness value vs Problem size

As per Figure 1, we can see that GA outperforms the rest of the algorithms. This can be justified as both RHC and SA are expected to underperform because the local minima in the Four Peaks problem have a wide base, which could lead to such algorithms being stuck at these minima.

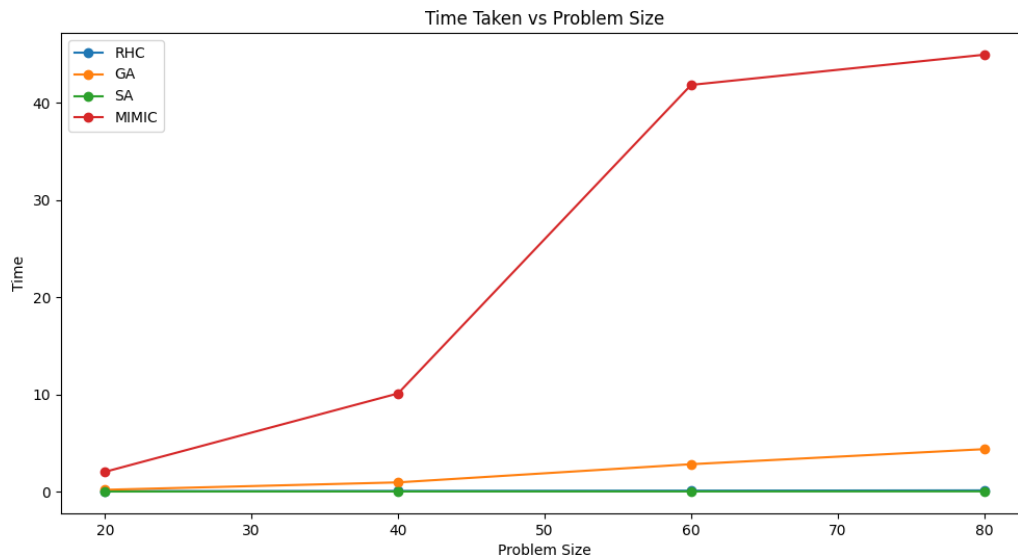


Figure 2. Wall time vs problem size for each algorithm

As expected, the MIMIC algorithm holds the highest execution time (we should expect to see this for all fitness functions).

Chosen algorithm for Four Peaks: **Genetic Algorithms** - Best fitness value & acceptable execution time.

## FlipFlop

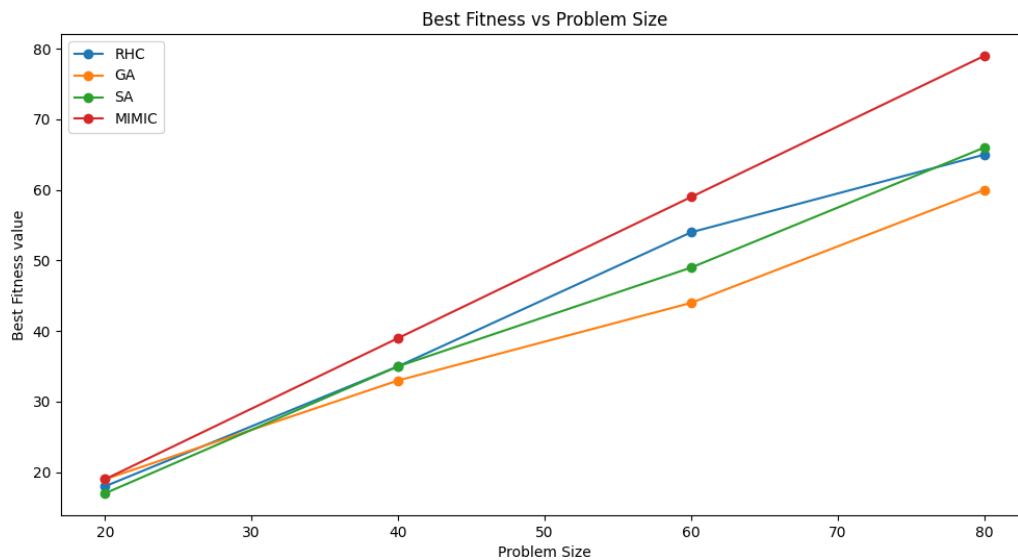


Figure 3. Fitness value vs Problem size on the FlipFlop problem

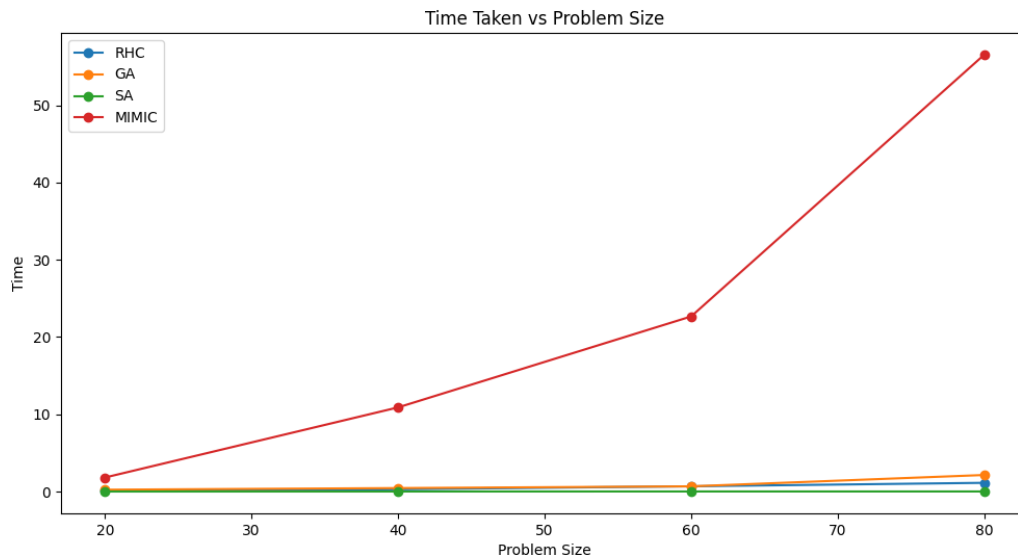


Figure 4. Wall time vs problem size for each algorithm

For this problem, the MIMIC algorithm is the clear winner, although with a high execution time, as it produces the highest fitness value. A justification for this is that MIMIC is able to keep information from earlier iterations to be used in future iterations down the line.

Chosen algorithm: **MIMIC**

## Count Ones

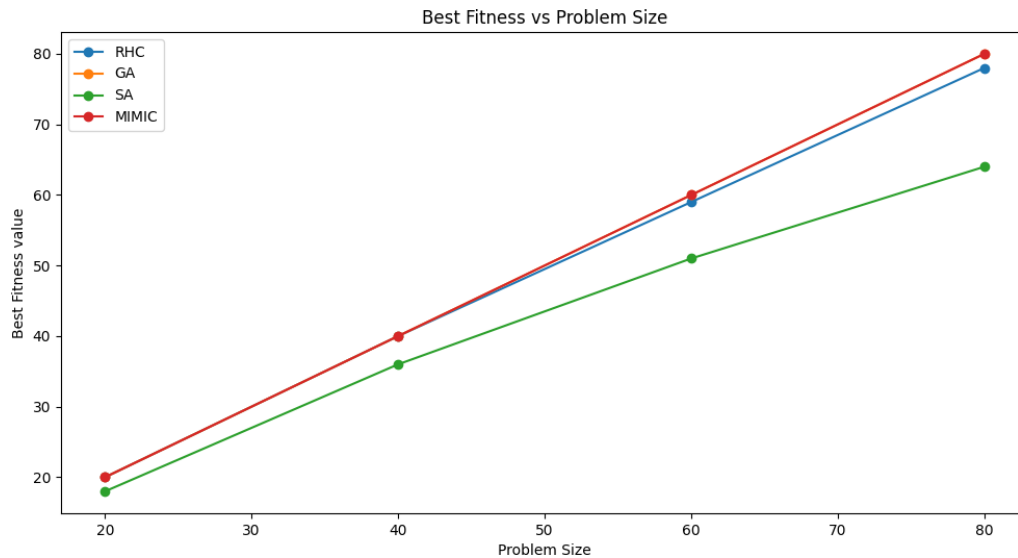


Figure 5. Fitness value vs Problem size on the FlipFlop problem

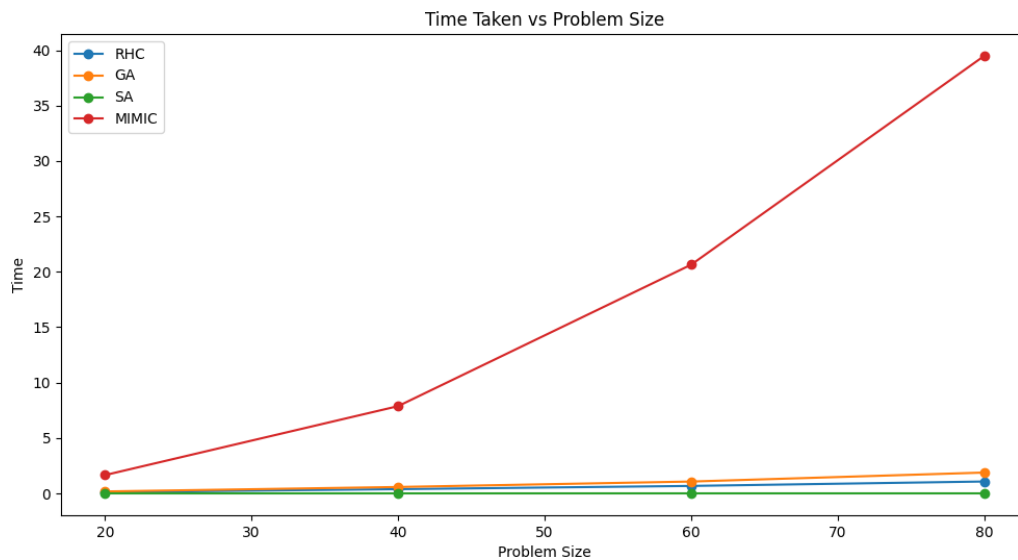


Figure 6. Wall time vs problem size for each algorithm

This problem is where we see the RHC performing well. This is because the Count Ones problem does not have local minima, meaning that it would likely converge to reach the optimal value. We would expect SA to have the same behavior but it needed more iterations before it could consume all the exploration capacity manifested in the Temperature parameter.

While MIMIC also performs well on this problem, **RHC** would be the algorithm of choice, due to its faster fitting time.



## Part 2: Neural Networks

In assignment 1, we designed a binary classification problem using the Titanic dataset. We trained a neural network using stochastic gradient descent to reach the optimal set of weights that corresponded to the best error rate. To keep the comparison fair, we choose the same parameters of the neural network used in assignment 1, with 2 hidden layers [40,40] and the ReLU activation function.

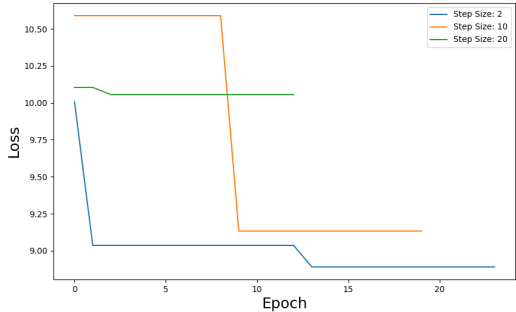
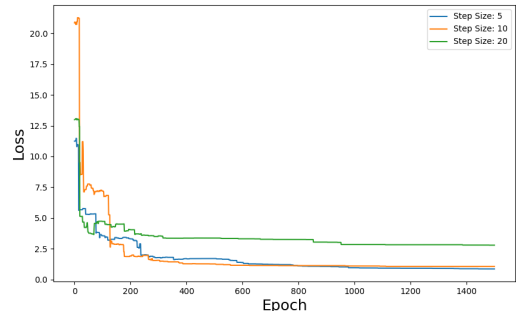
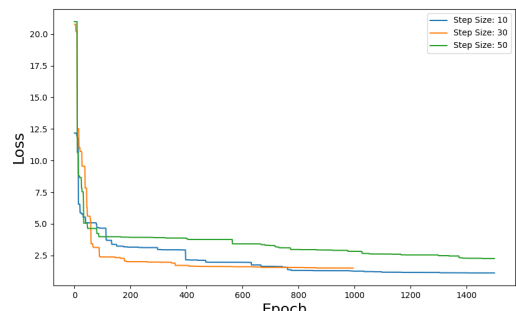
Optimization Algorithm	Loss vs Iterations
GA	 <p>Line plot showing Loss vs Epoch for the GA algorithm. The x-axis represents Epochs (0 to 20), and the y-axis represents Loss (9.00 to 10.50). Three lines are plotted: Step Size: 2 (blue), Step Size: 10 (orange), and Step Size: 20 (green). The Step Size: 2 line drops sharply from 10.00 to 9.00 by epoch 1 and remains stable. The Step Size: 10 line drops from 10.50 to 9.10 by epoch 8. The Step Size: 20 line drops from 10.25 to 10.10 by epoch 1 and remains stable.</p>
SA	 <p>Line plot showing Loss vs Epoch for the SA algorithm. The x-axis represents Epochs (0 to 1400), and the y-axis represents Loss (0.0 to 20.0). Three lines are plotted: Step Size: 5 (blue), Step Size: 10 (orange), and Step Size: 20 (green). All lines show a rapid initial drop in loss, followed by a slower decrease. The Step Size: 5 line reaches the lowest loss of approximately 1.0 by epoch 200. The Step Size: 10 line reaches approximately 1.0 by epoch 400. The Step Size: 20 line reaches approximately 2.5 by epoch 600.</p>
RHC	 <p>Line plot showing Loss vs Epoch for the RHC algorithm. The x-axis represents Epochs (0 to 1400), and the y-axis represents Loss (0.0 to 20.0). Three lines are plotted: Step Size: 10 (blue), Step Size: 30 (orange), and Step Size: 50 (green). All lines show a rapid initial drop in loss, followed by a slower decrease. The Step Size: 10 line reaches the lowest loss of approximately 1.0 by epoch 200. The Step Size: 30 line reaches approximately 1.0 by epoch 400. The Step Size: 50 line reaches approximately 2.5 by epoch 600.</p>

Table 1. Plots using each algorithm as an optimizer for our neural network



The plots in Table 1 show that all optimizers would help the neural network converge. The only concern is that if the learning rate is larger than required, the loss would likely diverge, as shown for the GA plot.

By experimenting with different effective learning rates, we are able to see all algorithms help the neural network to converge:

Optimizer	Testing Accuracy
GA	0.78
SA	0.76
RHC	0.75
Stochastic Gradient Descent (Backprop)	0.80

Neural networks' backpropagation are like any other optimization problem in an attempt to arrive at the optimal weights to solve the problem at hand. When we begin to solve these optimization problems using our random search algorithms for this section, we realize that they as well can more or less help the network converge on the dataset. That is why, it is expected to see the network converge to nearly the same level of classification accuracy on the Titanic dataset.

## Conclusion

In this assignment, we have explored the behavior of four randomized optimization algorithms. RHC is the simplest of them all, where it simply does basic hill climbing but with a randomized restart. SA is an advanced version of RHC but with an extra parameter T (Temperature) which balances between the exploitation and exploration cycle. GA creates new refined subsets of the data with “*elite*” traits that can help the algorithm make proper selections. MIMIC has a memory-like feature where it can make use of history across earlier iterations to move in better directions in further iterations.

There is no one-for-all algorithm; this means that the choice of algorithm has to depend on the kind of problem we are dealing with, and that is what was hopefully shown in this assignment. RHC requires simple problems, best with no local minima. SA is guaranteed to converge but can take a longer time to do so, depending on the amount of peaks in the problem. GA are generally powerful against non-linear optimization problems. MIMICs are powerful in most domains but they are quite expensive to run.

## References

<https://medium.com/@duoduoyunnini/introduction-implementation-and-comparison-of-four-randomized-optimization-algorithms-fc4d96f9feea>