**Problem Descriptions**

The three optimization problems I chose are OnesMax which is built-in with mlrose and two custom functions titled max\_half\_and\_end and max\_three\_zeros. A short description of these three is as follows:

* OnesMax – Find as many ones as possible. The best string would be all ones.
* max\_half\_and\_end – The first half of the bit string should be ones, and the final digit in the string should also be a one. Everything else in the second half should be a zero.
* max\_three\_zeros – This function attempts to maximize the number of substrings of “000” separated by ones in the given bitstring.

**Optimization Problems**

For each optimization problem listed below, I began by doing a grid search over various permutations of maximum iterations and maximum attempts. The values for both lists were 10, 50, 100, and 500.

**Max\_three\_zeros**

* **Genetic Algorithm –** This problem showed real strength with the genetic algorithm with a slight caveat. While the next two problems were nearly linear in their ascent towards an optimal solution, the genetic algorithm provided a huge jump in optimality for this problem within the first several iterations. The caveat here is that grid search suggested that this problem use 100 attempts and 500 iterations. The same values for the other problems may have provided better results, but would also likely have been computationally inefficient. Searching over mutation probabilities found .7 to be ideal, and ideal population seemed to switch between 100 and 200, so I selected 100. This problem also attained the highest normalized fitness score as the length of the problem grows. While the initial run requires over half of the 500 iterations to find an optimal value, optimizations allow the algorithm to find it in slightly over a fifth.

|  |  |  |
| --- | --- | --- |
| **A close up of a map  Description automatically generated** | **A close up of a map  Description automatically generated** | **A screenshot of a cell phone  Description automatically generated** |

* **Simulated Annealing – Lorem ipsum**
* **MIMIC – Lorem ipsum**
* **Random Hill Climbing – Lorem ipsum**

**OnesMax**

* **Genetic Algorithm –** The initial grid search found that the values of 10 attempts with 100 iterations produced the best fitness value. To test how other hyper parameters affected performance, I first tested [.1 .2 … .8, .9] for the mutation probability. This search found that a rate of .1 produced the highest average. Next, I tested against different population sizes which included [2, 10, 50, 100, 200, 500]. At a population of 50, the algorithm’s performance improves substantially, and it continues to improve as the problem’s length grows larger.
* **Simulated Annealing –** I begin by doing a grid search of all possible
* **MIMIC – Lorem ipsum**
* **Random Hill Climbing – Lorem ipsum**

**Max\_half\_and\_end**

* **Genetic Algorithm –** An initial grid search found values of 10 attempts and 100 iterations to be optimal. Testing against mutation probabilities found that .2 was optimal. Against multiple population sizes, similar to the OnesMax problem, more was better, so 500 was selected. After changing each of these optimizations, the algorithm performs slightly better than the initial run with a smaller population size and the default mutation probabilities. The optimum is found a few iterations earlier than without these optimizations.

|  |  |  |
| --- | --- | --- |
| **A screenshot of a cell phone  Description automatically generated** | **A screenshot of a cell phone  Description automatically generated** | **A screenshot of a cell phone  Description automatically generated** |

* **Simulated Annealing – Lorem ipsum**
* **MIMIC – Lorem ipsum**
* **Random Hill Climbing – Lorem ipsum**

**Neural Network**

**Overview**