Rohith Krishnan
CS4641-B
11 March, 2018

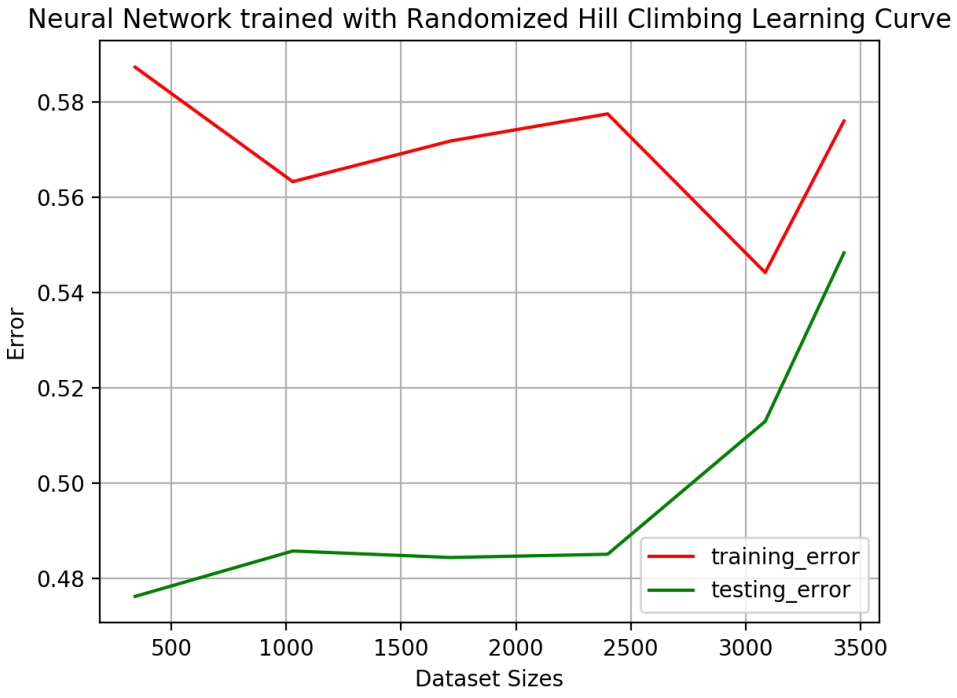
Randomized Optimization Analysis

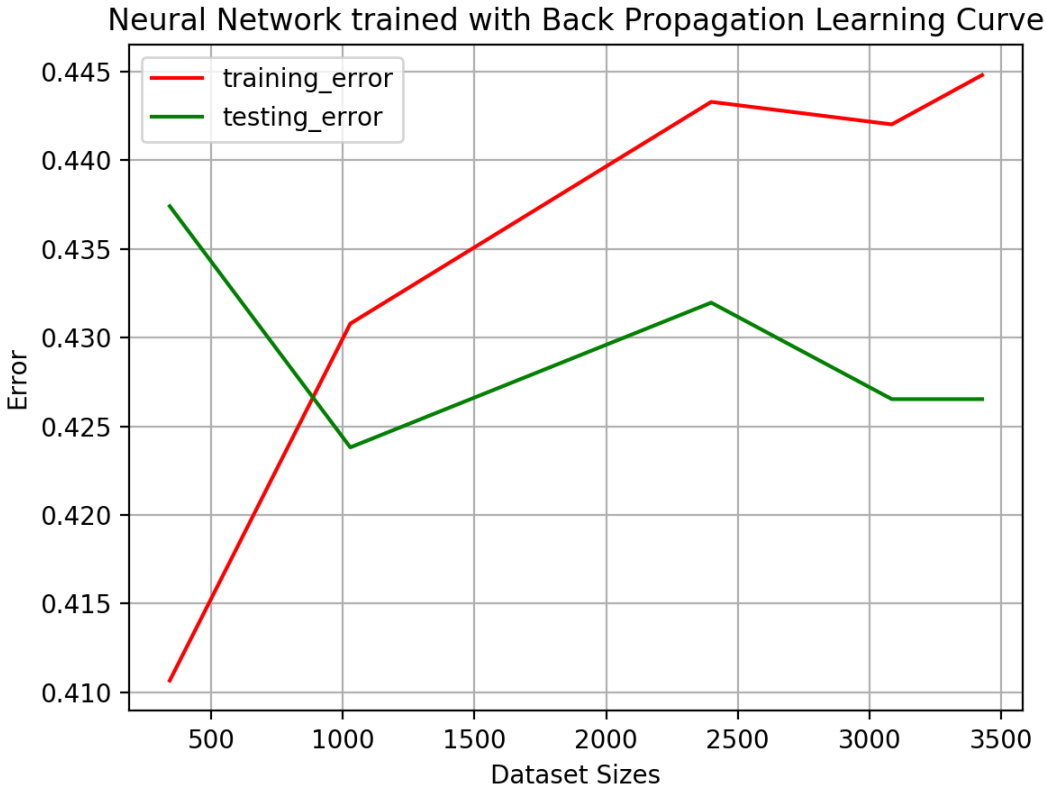
Classification Problem

Wine Quality- This dataset contains different physiochemical data about different types of Portuguese wine and includes quality scores for each wine on a scale from 0 to 10. The classification problem was to predict the quality score of the wine given features such as the alcohol percentage and acidity of the wine sample. The different features of the dataset were scaled between 0 and 1 to make the computation easier for the neural network while the dataset itself contained 12 features with 4899 examples. The different Randomized Optimization techniques used to find good weights for a neural network were compared to a neural network that used gradient descent backpropagation to update its weights. Each algorithm ran for 2500 iterations and the network itself was trained on increasing sizes of the training set.

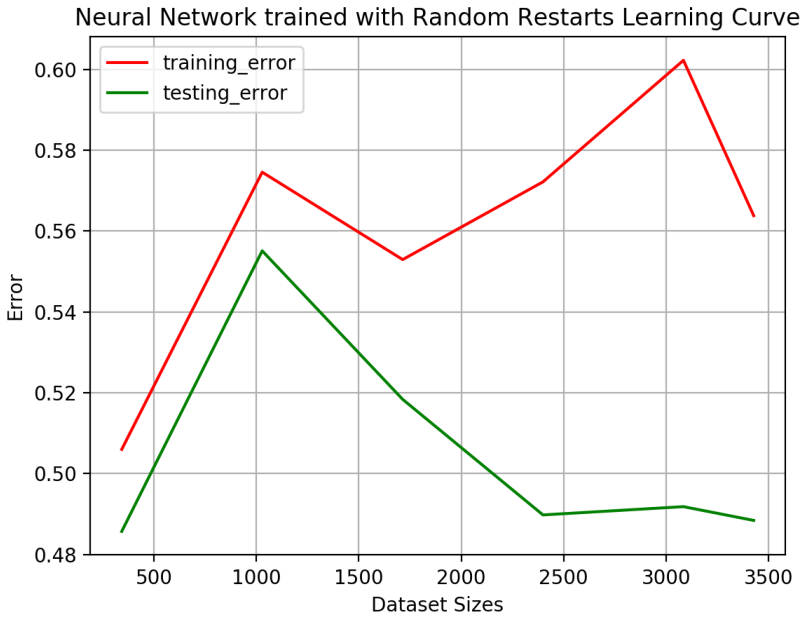
Randomized Hill Climbing

Randomized Hill Climbing is a greedy approach to optimization problems. It works by picking a random starting location in the heuristic space and finds neighbors close to that location in the space. It will then pick the best neighbor and will repeat until it reaches a point where all the neighbors to that point are worse than it. The learning curves of a neural network trained with back propagation and a neural network trained with randomized hill climbing are shown below.





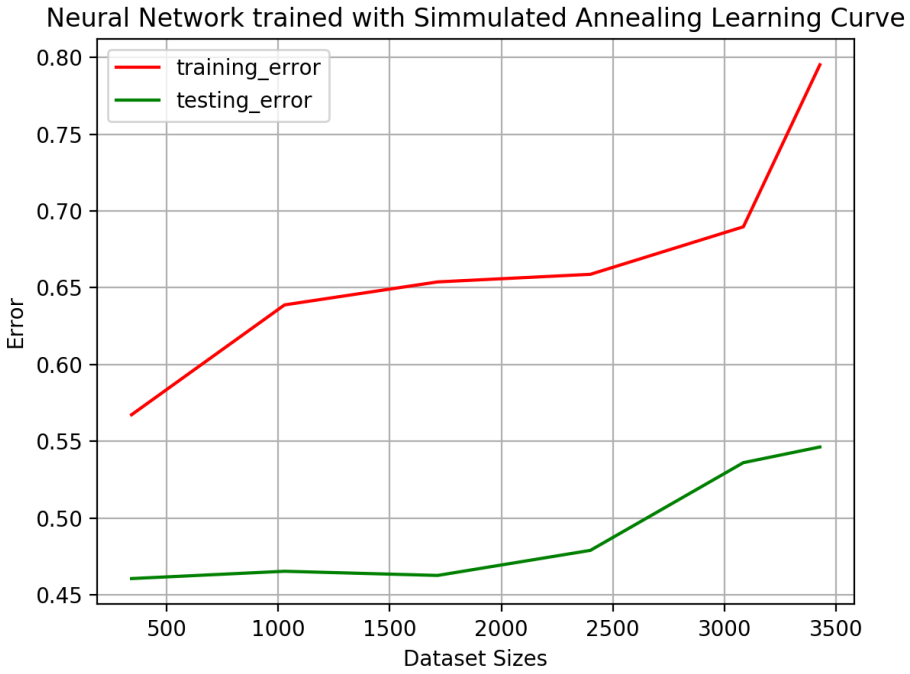
From these two graphs, we can see that a neural network trained using back propagation has a lower testing error rate than a neural network trained with randomized hill climbing. Also, the testing and training error for RHC started to increase as the size of the training set increased, indicating that the neural network was starting to under fit. One reason for RHC's relatively higher error rate is that the algorithm could have been stuck in some local optima in the space of possible weights for the neural network. This is highly likely since the weights of a neural network are continuous values which would result in a very large search space with many local optima. Since RHC randomly chooses a starting location, it can pick very unlucky points that only converge to local optimum and is more likely to do so in this optimization problem.



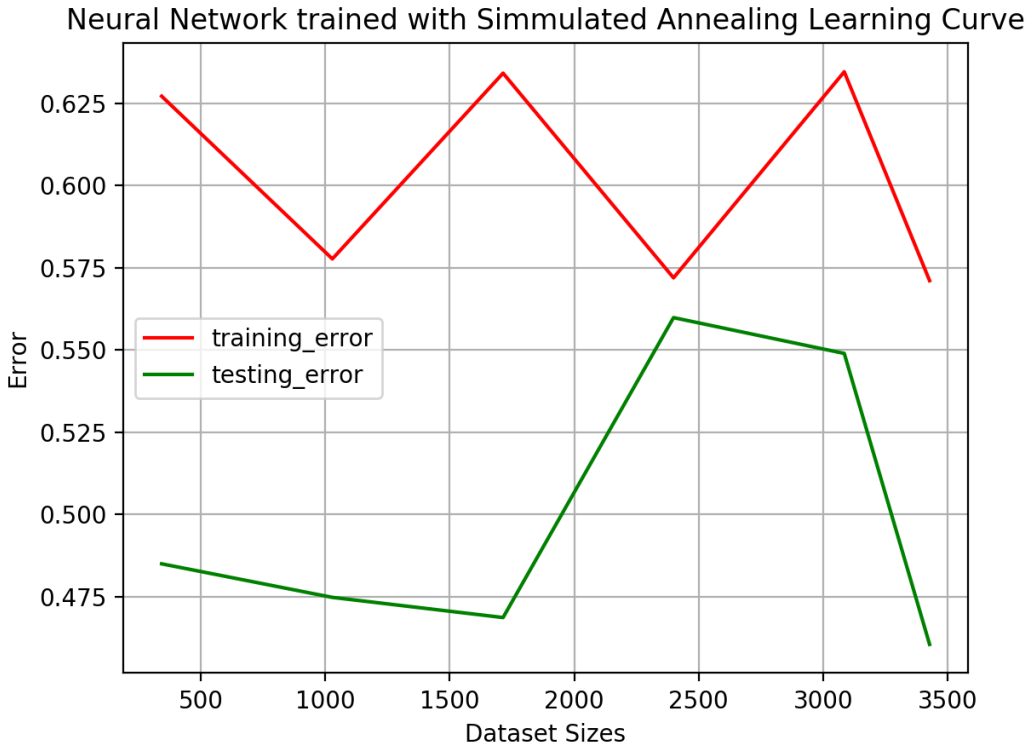
One way to overcome this issue is to have RHC randomly restart at a new point for a certain number of iterations or until there is no observable improvement. I ran the randomized hill climbing again with five restarts. As seen in the learning curve above, the training and test error rates started to decrease as the size of the training set increased. Overall random restarts helped the neural network have less error relative to the regular RHC algorithm. This improvement is attributed to how random restarts allows the hill climbing algorithm to search more of the space while there is still improvement based on the evaluation function.

Simulated Annealing

Simulated Annealing explores a lot more of the space by taking less random starting locations over time based on a schedule of decreasing "temperature". Temperature in this case is a probabilistic measure of selecting a random point in your search space or choosing the neighboring point in your search space based on your neighbor function. For this problem, I ran simulated annealing with a starting temperature of $1 * 10^{11}$ and a cooling factor of .95.



Based on the learning curve above, the neural network achieves a slightly better testing error rate than a network trained with random hill climbing but training and testing error still increase as more training samples are given which indicates under fitting. While simulated annealing does have slight improvements over random hill climbing algorithms, it still struggles in spaces that have many local optima. Also, the starting temperature and temperature change factor influence when simulated annealing will take random walks across the search space and how long it will take for the algorithm to converge to an optimal value. Running the simulated annealing algorithm again with a starting temperature of $1 * 10^{12}$ and a cooling factor of .85 allowed the algorithm to search more of the space as the neural network was able to attain a test error of less than 47.5% as more training samples were given to it. The training and testing errors decreased with more training examples which indicates that the neural net could generalize a bit better than the random hill climbing and back propagation training methods. The learning curve for this run of Simulated Annealing is shown below.

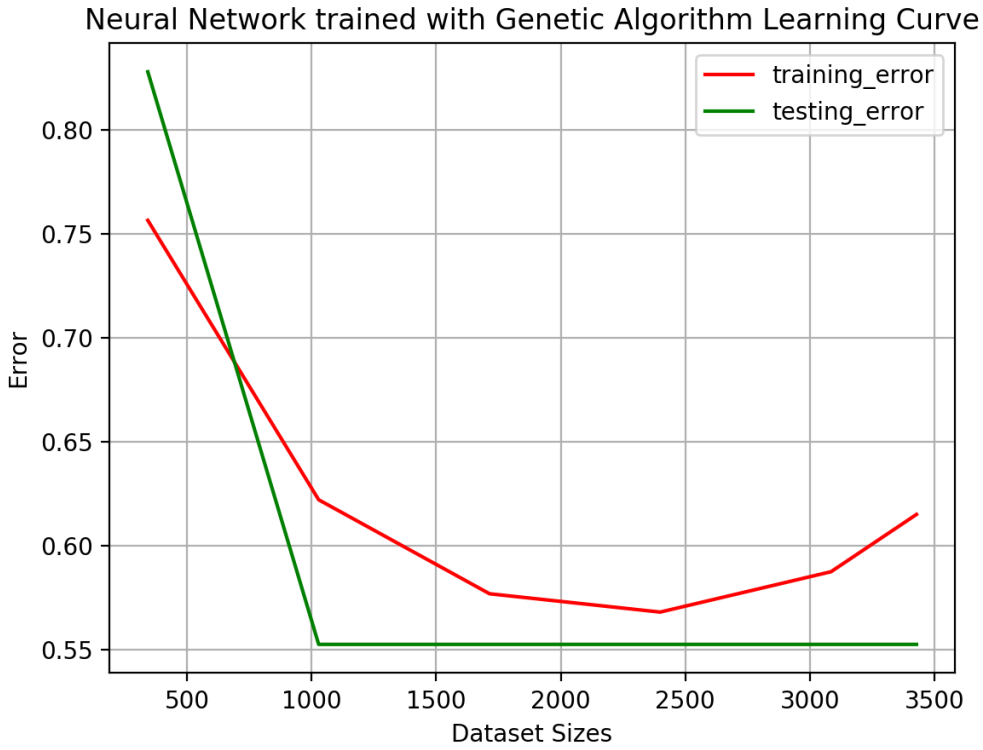


Genetic Algorithms

Genetic Algorithms are different from hill climbing and simulated annealing approaches because it starts from many points spread across the search space. It then uses different knowledge and heuristics gained from those starting points and reduces it search space by a larger amount each iteration. I ran a genetic algorithm on this problem with an initial population size of 100. At each iteration, the algorithm mutated 5 members of the population and crossed over 20 members.

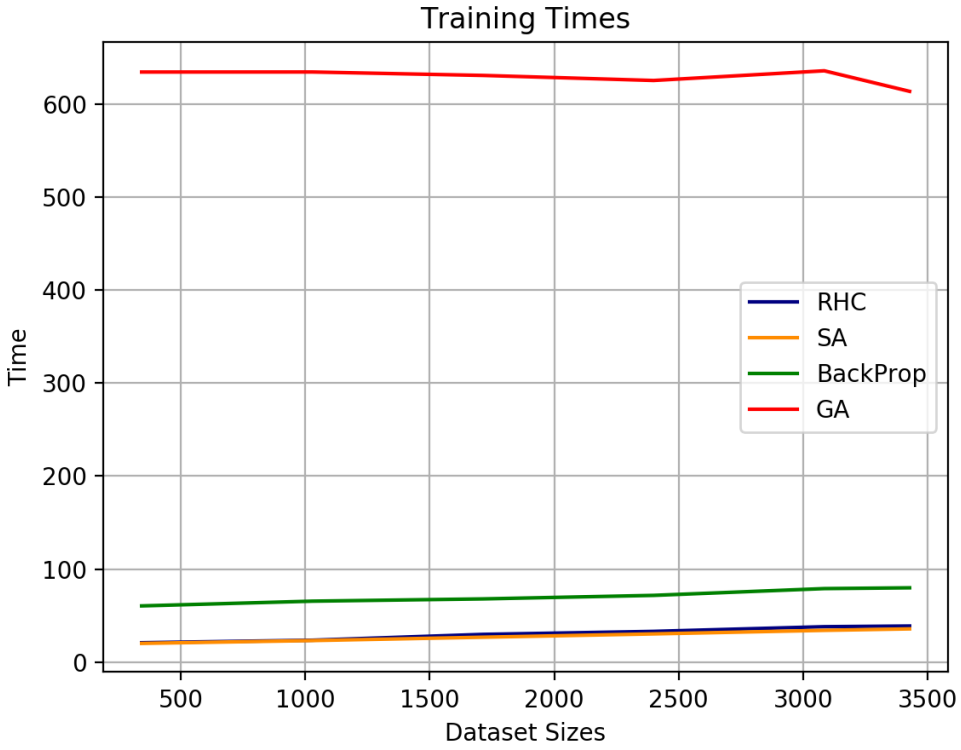
As seen in the learning curve below, the neural network starts out with a very high testing error but then converges to an error of 55%. The training error on the other hand is steadily decreasing with more training samples but starts to increase towards the end which indicates that the neural network is starting to over fit. Compared to the other algorithms, a Genetic Algorithm seemed to perform poorly as it had a higher average error rate. This performance could be caused by the fact that the cross-over and mutation functions are not well defined for this problem. Since the weights of a neural network are continuous values, the cross-over functions could be taking two members that are very similar to each other in terms of the heuristic function and could be making a new member that does not separate itself from the original population and does not help the algorithm converge to the best optima. Additionally, when the genetic algorithm changes the weights of a neural network even slightly, this would have a large impact on the output of the neural network and its accuracy. These small changes would also affect the fitness of the genetic algorithm in each iteration. Accuracy may have been improved by varying the

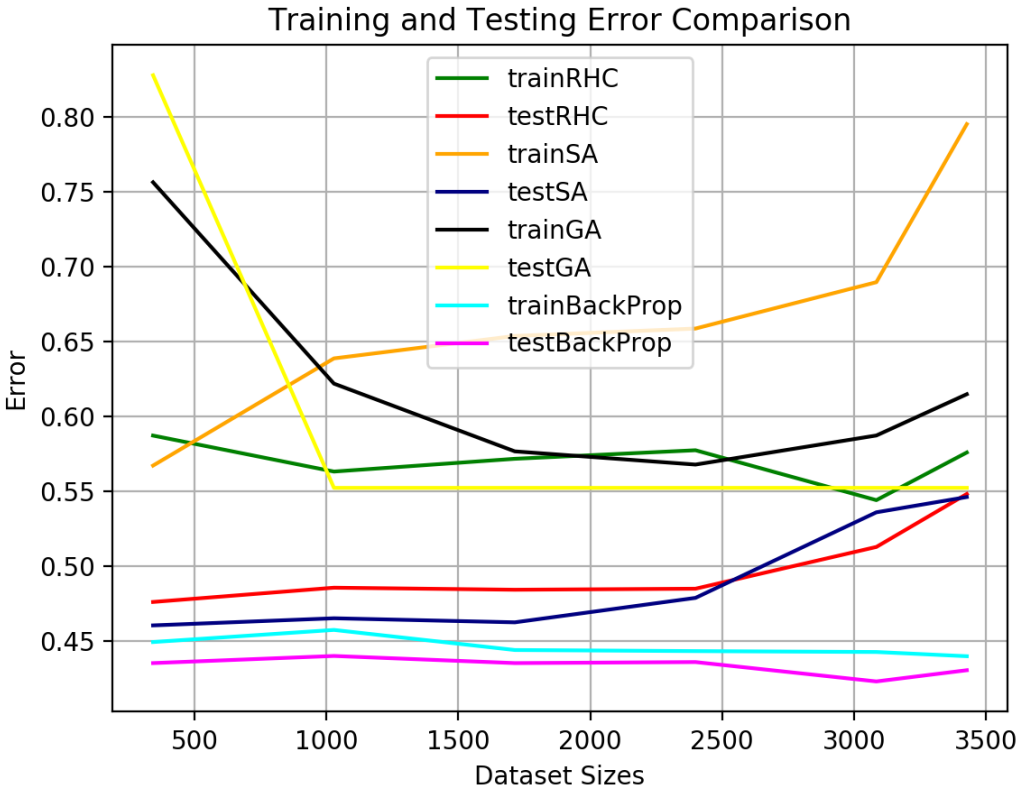
starting population and the number of members that were mutated or crossed over in every iteration, but the large training time for this algorithm made these tests difficult to complete.



Comparison

When comparing training times of the different algorithms, Genetic Algorithms took almost ten times the amount of time to train when compared to Simulated Annealing and Random Hill Climbing. When we also compare the training and testing error rates to the other two algorithms, we see that genetic algorithms produce testing less accuracy for the neural network than the other two approaches. The combination of long training time and relatively low accuracy shows how Genetic Algorithms are not well suited to optimizing the weights of a neural network. On the other hand, Simulated Annealing took about the same time to train as Randomized Hill Climbing, but had a lower testing error on average. The relatively lower testing error for the given training time shows that Simulated Annealing was the better randomized optimization algorithm to use for this problem. However, when compared to back propagation, Simulated Annealing still had a higher training and testing error rate even though it had a faster training time. I think Simulated Annealing could perform as well or even better than back propagation given a larger starting temperature, a lower cooling schedule, and more training iterations. The graphs comparing training times and training and testing error rates can be seen below.


Training Times
Time
Dataset Sizes
RHC
SA
BackProp
GA
0
100
200
300
400
500
600
500
1000
1500
2000
2500
3000
3500


Training and Testing Error Comparison
Error
Dataset Sizes
trainRHC
testRHC
trainSA
testSA
trainGA
testGA
trainBackProp
testBackProp
0.45
0.50
0.55
0.60
0.65
0.70
0.75
0.80
500
1000
1500
2000
2500
3000
3500

Flip Flop Optimization Problem

The Flip Flop evaluation function counts the number of times that a bit string alternates between 0 and 1 including the first bit. For example, the string '0101' would have a value of 4 and the string '1100' would have a value of 2. The optimal solution is to have a constantly alternating bit string that has a return value equal to the length of the bit string. I tested the four optimization algorithms on this problem with a bit string of length 120. I ran each algorithm fifty times.

Algorithm	Parameters	Average Fitness	Running time (seconds)
Random Hill Climbing	Iter: 20000	98.3	3.41
Simulated Annealing	T = 1E11, Cooling = .95 Iter: 20000	118.48	8.12
Genetic Algorithm	Starting Pop = 200, To Mate = 100, To Mutate = 20 Iter: 1000	103.56	6.10
MIMIC	Samples = 200, To Keep = 5 Iter: 1000	112.98	272.42

Simulated Annealing was the closest in producing the optimal solution when compared to the other three algorithms. Since the space of this problem is small and is based on discrete values, the local optima of the space are relatively close to each other. This means that a simulated annealing algorithm can explore the space more often than it chooses to stay at the current best point, preventing it from getting stuck at local optima. Random Hill Climbing gets stuck at local optima more often in this problem because it is not willing to look at worse solutions and only picks the best solution that is closest to its starting point. Genetic Algorithm performs slightly better than RHC in this problem but is still not able to converge to the optimal value because of its crossover function. Since the GA is using single point crossover, it is possibly losing parts of an optimal sequence and is combining two suboptimal parts together for the next generation of solutions. MIMIC on the other hand, requires much more time than the other three algorithms and would possibly need a lot more to reach the optimal solution. In terms of running time, Simulated Annealing was the best approach for this problem given its level of performance compared to the other three approaches.

Traveling Salesman Problem

The Traveling Salesman Problem involves finding the shortest path that visits all the nodes in a graph with N nodes. The problem is set up to have a specific number of vertices with random weighted edges. Each algorithm tries to maximize the inverse of the total length of any path that visits all the nodes in the graph. I ran this problem with 50 vertices in the graph and each algorithm was run 50 times.

Algorithm	Parameters	Average Fitness	Running Time (Seconds)
Random Hill Climbing	Iter: 20000	0.12083560482981874	4.138253077
Simulated Annealing	T = 1E11, Cooling = .95 Iter: 20000	0.12064100263732475	13.790267814999996
Genetic Algorithm	Starting Pop = 200, To Mate = 100, To Mutate = 20 Iter: 1000	0.14475265822030067	13.960412475999997
MIMIC	Samples = 200, To Keep = 5 Iter: 1000	0.1090595420331974	675.9824241560004

Genetic Algorithms had the best performance in terms of fitness. The space of possible solutions for this problem is much larger than the flip flop problem and there are many local optima in this problem. This would lead to a random hill climbing approach not reaching the best solution as it is more likely to get stuck at some local optima. For simulated annealing, the size of the space makes it much more difficult for it to converge to the global optima since it is only looking at one possible solution at a time. Genetic Algorithms can search more of the space since it is picking multiple start points for its initial population. It also tweaks and combines partial solutions through its mutation and crossover functions which makes it more likely for the algorithm to converge to the global optima of the fitness function. MIMIC again required a lot of time to run as it is constructing new probability distributions each time it reduces the space it is searching through. Since the space of this problem is very large, MIMIC would probably need many more iterations to reach the global optima, thus resulting in the poor results for this problem.

Four Peaks Problem

The Four Peaks evaluation function takes in a bitstring of length N and a position in the string T. The function has two global maxima when there are T + 1 leading 1s followed by all 0s or when there are T+1 leading 0s followed by all 1s. There are also two local maxima which occur when the string is all 1s or all 0s. The global maxima that the function could have would be 2N – T – 1. I ran all four optimization algorithms on this problem with N = 100 and T = 10. Each algorithm ran for 50 iterations. The results of these trails can be seen below.

Algorithm	Parameters	Average Fitness	% of Optimal Solution	Running Time (Seconds)
Random Hill Climbing	Iter: 20000	100.0	0%	3.0292144720000005
Simulated Annealing	T = 1E11, Cooling = .95 Iter: 20000	119.58	22%	3.0292144720000005
Genetic Algorithms	Starting Pop = 200, To Mate = 100, To Mutate = 20 Iter: 1000	97.32	0%	3.782430922
MIMIC	Samples = 200, To Keep = 5 Iter: 1000	123.14	26%	192.95923373599993

For this problem, Random Hill Climbing and Genetic Algorithms performed very poorly as they failed to reach the optimal solution across all 50 trials. The Random Hill Climbing approach seemed to get stuck in the two local optima on every trial. Because the algorithm randomly picks starting locations and greedily searches for the best neighbor, it is very unlikely that it will reach the global optima if it was given more iterations. The poor performance of the genetic algorithm can be attributed to the cross over function not being well suited for this problem. Since the algorithm was using single point crossover, it could have been selecting crossover points that included the point T used in evaluation. This would result in creating new members that have been created from partial suboptimal solutions, thus not advancing the algorithm to the global optima.

MIMIC had the best performance on this problem in terms of how often it reached the optimal solution. In each iteration, MIMIC is forming a probability distribution of the most probable locations of local maxima in the search space. It then uses this distribution in the next iteration to determine the best location to search next. By repeating this process multiple times, the suboptimal solutions are removed as better optima are found, making it more likely for the algorithm to converge to a single global optimum. MIMIC seemed to perform better in this space than the previous two problems because the optimal values were well defined. Unlike the traveling salesman problem which has a very large search space and multiple local optima, this problem only had two local optima and two global optima and a much smaller search space. This problem space made creating a probability distribution much simpler and allowed the algorithm to eliminate more suboptimal choices as it converged to one of the global optima. Like the other two problems, the running time of MIMIC was around ten to a hundred times longer than the other three optimization algorithms due to the amount of time it takes to form a probability distribution of all the attributes that define the underlying space.