Randomized Optimization Assignment

Jun Wang
CS7641 – Assignment 2
Jwang3316@gatech.edu

Abstract

Four randomized optimization problems (Random Hill Climbing, Simulated Annealing, Genetic Algorithm, and MIMIC) are implemented on three different Discrete Optimization Problems and their properties are highlighted. After this step, the first three algorithms are implemented on the neural network weights to optimize the network to make the best predictions given the dataset, which is used at the previous assignment.

1. Introduction to Problems

1.1. N-Queens Problem

N-Queens Problem requires placing N-Queens on a NxN chess board with the purpose of no two queens could attack each other. This problem is an NP-Complete problem that has a non-convex optimization surface. One change in the placement of a queen could change the fitness function greatly.

This problem is a great start for the comparison of randomized optimization algorithms because of these features. I believe it will bring out the weaknesses of them. I used N=64, which makes it very difficult to solve by the solvers completely so that they all will try to minimize the number of queens threatening each other.

1.2. Flip-flop Problem

Flip-flop problem consists of consecutive bits and the fitness function is the sum of consecutive different bits such that 1010 => 3 and 1110=> 1. Even though this problem is very easy to solve for us, it is a nice challenge for random optimization algorithms because the bits are only dependent to the two bit next to them. Using this problem, we will see how different the algorithms will behave under this kind of dependency.

1.3. Knapsack Problem

Knapsack is one of the most popular problems in optimization world, which aims to fill a bag with weight capacity using items and maximize the total value of the items in the bag. This is another NP-Complete problem. It has many many local optimas and can be a good comparison for the optimization algorithms.

2. N-Queens Problem

In this problem, I have used a neighbor search parameter of 100 at each algorithm so none of them benefited from a larger search at each iteration.

Because I have used 64 queen setup in this problem the total number of queen attacks to avoid is 2080, which is the max of the fitness function.

2.1. Random Hill Climbing

Hill Climbing algorithm is a great tool for improving the fitness function only by testing neighbors and moving to the better direction. But, it cannot overcome local optima and got stuck on them. So, a simple solution to this is just to start the search over at randomized locations, which is called Random Hill Climbing.

This algorithm is the first one I have used and it is good to see how other algorithms behave comparing to RHC.

I have used 100 random restarts with 100 max_attempt to get a better neighbor.

2.1.1 Time/Iteration Analysis

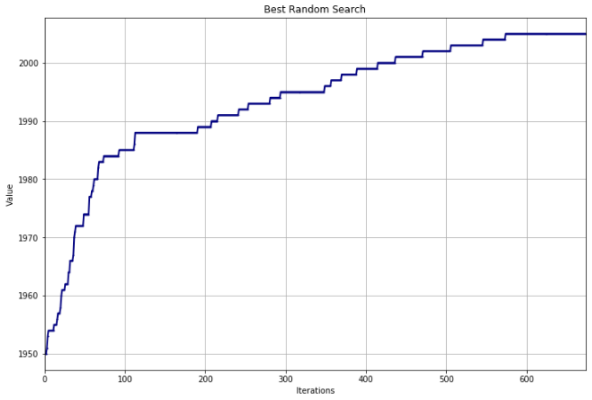
	precision	recall	f1-score	support
0	0.58	0.92	0.71	315
1	0.84	0.71	0.77	403
2	0.61	0.33	0.43	282
micro avg	0.67	0.67	0.67	1000
macro avg	0.67	0.66	0.64	1000
weighted avg	0.69	0.67	0.65	1000
samples avg	0.67	0.67	0.67	1000

The algorithm took a total amount of 902 seconds in this problem, which makes 9 seconds on average for each and 55856 iterations in total, 558 steps for each iteration, whereas the best search finds its local optima in more than 650 steps. It gets stuck later than average as expected.

2.1.2 Optimization Value Analysis

The average hill climbing algorithm in the set of 100 gets 1999, which is reasonably high and the best of them is 2005. Interestingly, the difference between the average and the maximum is so low in this problem. This is because the dimensions of the problem is too large and probably it can find a way to go up until a level and after some improvements, it can't get lucky because so many queens are present and it hits a local optima.

The best random search algorithm's learning curve over iterations



It can be seen that in this problem, there are long plateaus where hill climbing gets over after trying long enough. It finds the only way to go up and gets stuck in the local optima in the end.

2.2. Simulated Annealing

At simulated Annealing algorithm, I have used two different Temperature Decay functions (Geometric and Exponential) and 9 different initial temperature parameters (1, 10, 50, 100, 250, 500, 1000, 2500, 5000).

A total of 18 different simulated annealing algorithms are run in this problem.

2.2.1 Time/Iteration Analysis

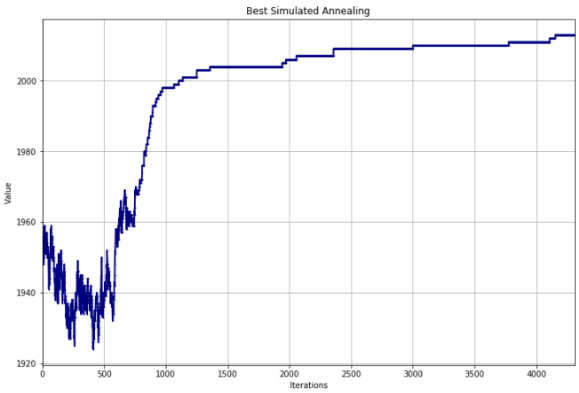
In the GridSearch design illustrated above, it is expected to have longer fitting times when initial temperature is higher. But the experiments didn't give such a result. Interestingly, they all take a reasonable amount of time which is in a factor of 2 (minimum is 7 seconds and maximum is 15).

2.2.2 Optimization Value Analysis

The average simulated annealing algorithm got a fitness function of 2010, which is still far from the global optima. The best of the algorithms obtained 2013, which does not seem very different than average.

The setup that gave the best results uses a Geometric Decay algorithm with an initial start of 2500. Even though it got a better result than 5000 initial temperature, I think this comes from randomization and the larger temperatures will give a better (not worse) result.

The best simulated annealing algorithm's learning curve over iterations



It can be seen that the iterations below 1000, there are significant oscillations. It happens where the temperature is relatively high. And after iteration 1500, it looks quite similar to the hill climbing algorithm as mentioned in the lecture.

2.3. Genetic Algorithm

At genetic algorithm, I have used three different population sizes (50, 200, 500) and three different mutation rates (0.1, 0.25, 0.5).

A total of 9 different genetic algorithms are run in this problem.

2.3.1 Time/Iteration Analysis

In the GridSearch design illustrated above, it is expected to have longer convergence times when the

population size is higher. After experiment, this has proven correct and in this problem, the time spent is linearly dependent on the population size.

Population Size	
50	49.350818
200	192.022607
500	536.895150

Comparing to the first two algorithms, it is obvious that Genetic Algorithm spends much more time to find the best value it can.

2.3.2 Optimization Value Analysis

GA could get an average result of 2006 and maximum of 2012, which is as good as Simulated Annealing. But, it can be seen that the difference between the average and maximum differs more in GA comparing to SA.. The mean values of parameters are:

Mutation Rate		Population Size	
0.10	2006.666667	50	2002.000000
0.25	2007.000000	200	2008.333333
0.50	2007.000000	500	2010.333333

Population size seems to be significant and mutation rate seems to be insignificant on affecting the best fitness value reached.

2.4. MIMIC

At MIMIC algorithm, I have used three different population sizes (50, 200, 500) and three different percentage kept parameters (0.25, 0.5, 0.75).

A total of 9 different MIMIC algorithm are run in this problem.

2.4.1 Time/Iteration Analysis

Even though it takes around 4x more time to run the MIMIC comparing to Genetic Algorithm, the time spent changes linearly with regard to the population size.

MIMIC takes more than 4x time comparing to GA, which takes the most of the time among first 3 algorithms.

2.4.2 Optimization Value Analysis

MIMIC had a poor performance on N-Queens problem, which is worse than RHC. On average, the MIMIC algorithm obtained 1972.5 and maximum of 1989, which is the lowest maximum among 4 algorithms. But, it can be seen that the population size has a significant effect on the fitness value.

Population Size	
50	1958.333333
200	1974.000000
500	1985.333333

2.5. Algorithm Comparison

	RHC	SA	GA	MIMIC
Mean Score	1999.4	2010	2006.8	1989
Max Score	2005	2013	2012	1972
Mean Time (s)	9	11.8	259.4	1057

In this problem, two algorithms seem to bring out the best results. The next problem's comparison highlights Simulated Annealing better so I will explain why **Genetic Algorithm** worked good in this setup.

GA works in two ways: mutation and combination. Mutations help the fitness function as a simple search as in hill climbing but combinations are complex. In the N-Queens set-up, two different algorithms could set the queens very different than each other. But the survivor algorithms have to do good job in some amount of the queens. When combination comes in, it takes the parts where two different good algorithm works and combine them in this setup and it works unlike Flip-flop problem.

3. Flip-flop Problem

I have used 500 bits in this problem to make it considerably difficult for randomized optimization algorithms. In this context, the max fitness function will be 499.

In this problem, I have used a neighbor search parameter of 100 at each algorithm so none of them benefited from a larger search at each iteration.

3.1. Random Hill Climbing

I have used 100 random restarts with 100 max_attempt to get a better neighbor.

3.1.1 Time/Iteration Analysis

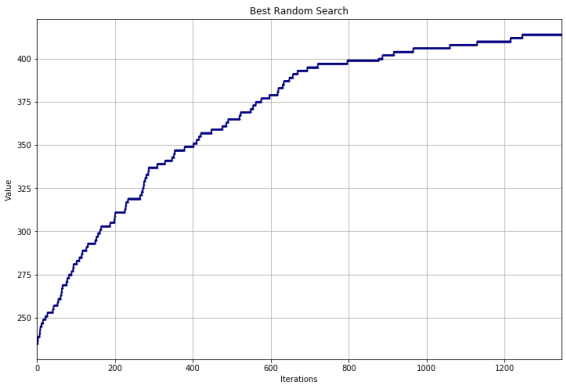
The algorithm took a total amount of 560 seconds, which makes 5.6 seconds on average for each and 93772 iterations in total, which makes around 938 steps for each iteration.

The best value took 1346 iterations, which shows that it got stuck much later than other hill climbing algorithms.

3.1.2 Optimization Value Analysis

The average hill climbing algorithm in the set of 100 gets 392, which is again reasonably high and the best of them is 414. In this setup, it can be seen that the Flip-flop problem seems to have many local optima but the levels of these local optima are not that low.

The best random search algorithm's learning curve over iterations



As it can be seen in the graph, the fitness function just goes up and where it can't, it gets stuck over there.

3.2. Simulated Annealing

At simulated Annealing algorithm, I have used two different Temperature Decay functions (Geometric and Exponential) and 5 different initial temperature parameters (1, 10, 50, 100, 250).

A total of 10 different simulated annealing algorithms are run in this problem.

3.2.1 Time/Iteration Analysis

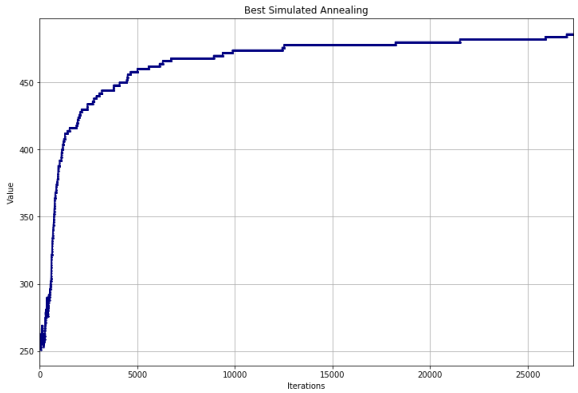
In the GridSearch design illustrated above, it is expected to have longer fitting times when initial temperature is higher. After the run time though, the experiment gave quite close convergence times for 1 to 100, whereas for 250 it took almost 3 times of the average. Comparing to the hill_climbing, every single iteration took 10x more time, which is still considerably low, given the difference in the fitness function.

3.2.2 Optimization Value Analysis

The average simulated annealing algorithm got an impressing 482.7 fitness function, which is quite close to the global optima. The best of the algorithms obtained 486, which does not seem very different than average, but actually it is a pretty large difference considering the amount of local optima in this problem setup at numbers close to the global optima.

The setup that gave the best results uses a Geometric Decay algorithm with an initial start of 250, which shows that starting from a larger initial temperature gives an significant advantage to the simulated annealing algorithm even though it increases the time of convergence.

The best simulated annealing algorithm's learning curve over iterations



It can be seen that the iterations that takes the simulated annealing to maximum is over 25000, which is almost 20 times of average hill climbing and this is because it doesn't stop on downhills depending on the current temperature level.

3.3. Genetic Algorithm

At genetic algorithm, I have used three different population sizes (20, 50, 100) and three different mutation rates (0.1, 0.25, 0.5).

A total of 9 different genetic algorithms are run in this problem.

3.3.1 Time/Iteration Analysis

In the GridSearch design illustrated above, it is expected to have longer convergence times when the

population size is higher. After experiment, this has proven correct but less than what I have expected. 100 population took almost 2x time comparing to the 20 population setup on average.

And the time of each genetic algorithm is interestingly very close to the random hill climbing instances in this problem. I think it is partly because of the poor performance of the Genetic Algorithm in this problem. It can't keep improving on this because it doesn't just go over different local optima and tries to combine them after mutating and this doesn't fit to the Flip-flop problem. It can be seen in the Simulated Annealing graph that it takes time to make improvement after a level and Genetic Algorithm doesn't have that sort of patience like SA.

3.3.2 Optimization Value Analysis

GA could get an average result of 409 and maximum of 428, which is quite poor, even when RHC can get 414 as maximum. When analyzed the means of population sizes, it can be seen that larger population size helps but not too much in this problem and so low or so high mutation rates are not preferable.

3.4. MIMIC

At MIMIC algorithm, I have used three different population sizes (20, 50, 100) and three different percentage kept parameters (0.25, 0.5, 0.75).

A total of 9 different MIMIC algorithms are run in this problem.

3.4.1 Time/Iteration Analysis

It can be seen that larger populations affect the convergence time a lot more significantly comparing to the Genetic algorithm and I believe this is caused because of the complex steps such as creating probability distribution in the MIMIC algorithm. In Genetic Algorithm, on the other hand, the step is only mutating and combining, which is much simpler.

Each MIMIC algorithm took 67 seconds which is much higher comparing to any other algorithm.

3.4.2 Optimization Value Analysis

MIMIC had a poor performance on Flip-flop problem, as well, even worse than RHC. On average, the MIMIC algorithm obtained 322.1 and maximum of 375, which is the lowest maximum among 4 algorithms. But, it can be seen that the population size has a significant effect on the fitness value as

Population Size	
20	286.333333
50	315.666667
100	364.333333

3.5 Algorithm Comparison

	RHC	SA	GA	MIMIC
Mean Score	392	482.7	409.1	322.1
Max Score	414	486	428	375
Mean Time(s)	5.6	60	5.05	67

It can be seen that the most successful algorithm is clearly **Simulated Annealing** in the FlipFlop problem. The advantage of this algorithm in this problem is that during the search after a slight drop in the fitness function caused by one bit (which can only change the fitness function +/-2) there is a potential of increase by just alternating the consecutive bits and Simulated Annealing does this the best.

4. Knapsack Problem

In this problem, I have used a neighbor search parameter of 100 at each algorithm so none of them benefited from a larger search at each iteration.

I have used 200 items, where each has a uniform random weight between 10 and 40. Also a uniform random value between 20 and 30. Maximum weight percentage is set to 50% so almost half of the items will be selected.

The reason I have used 200 items is to make things complicated for randomized optimization algorithms and see their behavior under this circumstances.

4.1. Random Hill Climbing

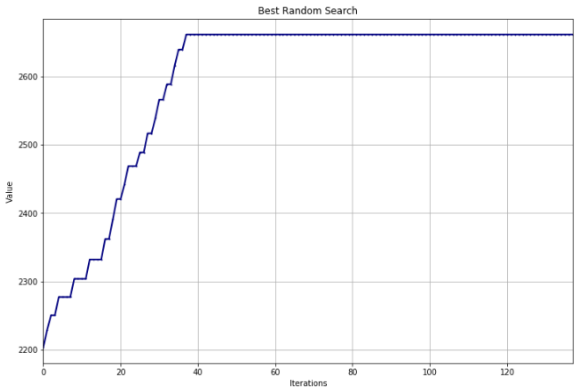
I have used 100 random restarts with 100 max_attempt to get a better neighbor.

4.1.1 Time/Iteration Analysis

The algorithm took a total amount of 7 seconds (0.07 each) in this problem to solve 100 hill climbing algorithms. This is caused because when weights are over the limit, the fitness function is 0 and because of that it is so easy to get stuck in the local optima.

4.1.2 Optimization Value Analysis

The average hill climbing algorithm in the set of 100 gets 1573, which is really low comparing other algorithms' results and the best of them is 2661. The randomization made a great effect on the result in this problem unlike others. This is because the algorithm got a really good start via randomization and improved it as it can be seen in the graph below.



It can be seen that the algorithm's initial fitness is just above 2200 and hill climbing brings it over 2600. It is quite amazing to get such a good initial value but this comes from the repeated randomization.

4.2. Simulated Annealing

At Simulated Annealing algorithm, I have used two different Temperature Decay functions (Geometric and Exponential) and 9 different initial temperature parameters (1, 10, 50, 100, 250, 500, 1000, 2500, 5000).

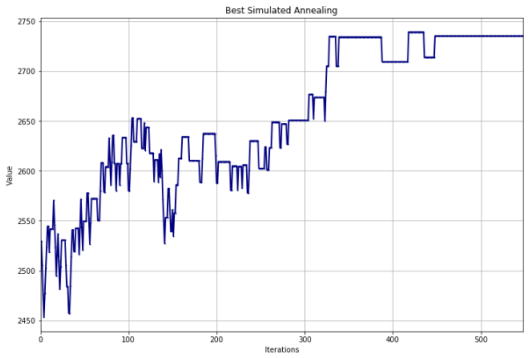
A total of 18 different simulated annealing algorithms are run in this problem.

4.2.1 Time/Iteration Analysis

Simulated Annealing took incredibly short amount of time like Hill Climbing algorithm. Each run took 0.07 seconds. This shows that it is much more difficult to get out of the local optima in Knapsack Problem, unlike Flip-flop problem.

4.2.2 Optimization Value Analysis

The average simulated annealing algorithm got a fitness function of 2619, which is still far from the global optima. The best of the algorithms obtained 2735, which does not seem very different than average.
The setup that gave the best results uses a Exponential Decay algorithm with an initial start of 50. In this problem, more temperature didn't affect positively and I think it is because the problem returns 0 when over-weighted and higher temperatures fall into this trap more often, because they change more into the negative sides.
The best simulated annealing algorithm's learning curve over iterations



It can be seen that Simulated Annealing can bring up lower initial fitness results to higher values because of its high-temperature behavior unlike Hill Climbing.

4.3. Genetic Algorithm

At genetic algorithm, I have used three different population sizes (500, 100) and three different mutation rates (0.1, 0.25, 0.5).

A total of 6 different genetic algorithms are run in this problem.

4.3.1 Time/Iteration Analysis

This problem is a very large problem so I expected the time spent to be much larger comparing to others in Genetic Algorithm. But, the results gave quite moderate convergence times, even with this large population sizes contrary to my expectation.

Population Size	
500	50.846998
1000	97.581183

Comparing to the first two algorithms, it obviously takes more time but Genetic Algorithm didn't spend a

lot more time given the performance improvement it obtained at this problem.

4.3.2 Optimization Value Analysis

GA could get an average result of 3254 and maximum of 3256, which is much better than the first two algorithms. The effect of population size can't be seen in this level because I have used really large two population sizes.

```
Population Size
500     3253.576229
1000    3256.218833
```

4.4. MIMIC

At MIMIC algorithm, I have used three different population sizes (500, 1000) and three different percentage kept parameters (0.1, 0.25, 0.5).

A total of 6 different MIMIC algorithm are run in this problem.

4.4.1 Time/Iteration Analysis

Comparing to other algorithms, MIMIC took longer time than Genetic Algorithm but the difference is not huge again.

```
Population Size
500     138.366355
1000    267.569381
```

It can be seen that the mean of convergence time spent on different population sizes doesn't increase more than linearly.

4.4.2 Optimization Value Analysis

MIMIC had its best performance in Knapsack problem, which is quite close to the Genetic Algorithm. Its population size affects the value in a great deal comparing to the Genetic Algorithm.

```
Population Size
500     3204.339830
1000    3244.854638
```

This kind of shows it still has potential if we increase the population size, it could get higher.

4.5. Algorithm Comparison

	RHC	SA	GA	MIMIC
Mean Score	1573	2619	3254	3224
Max Score	2661	2735	3256	3251
Mean Time (s)	0.7	0.6	61	202

In this problem, two algorithms seem to bring out the best results: Genetic Algorithm and MIMIC. **MIMIC** has the biggest performance difference comparing to other problems, though.

In Knapsack problem, some items are too good to leave outside, MIMIC just produces more samples from the density estimator which is constructed using the successful states. So, this problem fits to MIMIC and Genetic Algorithm quite well.

5. Neural Network Optimization

I have used the wine dataset that I have used in Assignment 1. The metric I have used was f1-weighted. So, I have used the f1-weighted metric in this part to evaluate the algorithms.

The Wine Dataset

Algorithm	Accuracy	f1-weighted
Decision Trees	0.9040	0.9035
Neural Network	0.9440	0.9432
AdaBoost	0.9230	0.9219
SVM	0.9340	0.9333
KNN	0.9280	0.9274

Different algorithms got these results on this wine dataset (testing).

In this assignment, I have used random optimization algorithms on 5-cross validation and used the best parameters on training and testing dataset to get the metrics.

5.1. Random Hill Climbing

First algorithm I have tried on the neural network weights optimization part is RHC gave really poor results at both training and testing dataset.

The grid-search I have tried was just on restarts (5, 25, 50) and learning rate (0.01, 0.001. All of the results (mean cross validation test scores) are super similar to each other (all between 0.2437 to 0.2483).

When observe the difference between training and testing results;

Training:

	precision	recall	f1-score	support
0	0.34	0.98	0.51	1262
1	0.81	0.01	0.02	1610
2	0.58	0.19	0.28	1128
micro avg	0.37	0.37	0.37	4000
macro avg	0.58	0.39	0.27	4000
weighted avg	0.60	0.37	0.25	4000
samples avg	0.37	0.37	0.37	4000

Testing:

	precision	recall	f1-score	support
0	0.34	0.98	0.51	315
1	0.86	0.01	0.03	403
2	0.73	0.21	0.33	282
micro avg	0.38	0.38	0.38	1000
macro avg	0.64	0.40	0.29	1000
weighted avg	0.66	0.38	0.26	1000
samples avg	0.38	0.38	0.38	1000

As it can be seen from these two tables, the network is clearly underfits to the data. The random hill climbing gets stuck at a local minima and can't move forward as expected, which results in this underfitting problem.

5.2. Simulated Annealing

Simulated Annealing has the clear advantage of being kind of immune to the local optima. So that it seems to have a great advantage comparing to Random Hill Climbing. So that I made a larger grid search with learning rate values (0.0001, 0.001, 0.0025, 0.005, 0.01) and an Exponential Decay function with initial temperature values (1, 10, 25, 50, 100, 10000).

As in Random Hill Climbing, the results didn't differ almost at all among different parameters. But, surprisingly to me, the Simulated Annealing algorithm gave a spectacular result on optimizing the neural network weights.

Training:

Testing:

	precision	recall	f1-score	support
0	0.90	0.92	0.91	315
1	0.91	0.92	0.91	403
2	0.92	0.88	0.90	282
micro avg	0.91	0.91	0.91	1000
macro avg	0.91	0.91	0.91	1000
weighted avg	0.91	0.91	0.91	1000
samples avg	0.91	0.91	0.91	1000

Simulated Annealing got 0.91 f1-score, which is pretty close comparing to 0.94 f1-weighted score obtained in Assignment 1 (using Adam optimizer). The weights given by Simulated Annealing seems to be underfitting because training has lower value then testing. Despite that, the results are pretty impressive.

5.3. Genetic Algorithm

Since Genetic Algorithm will mutate and combine different relatively good weights, it seems promising. When two different good states match, they can produce a better one.

So, I have used a large population size (200) and 2 different mutation probability (0.1, 0.25) and 2 different learning rate (0.01, 0.1). These values are selected to balance the performance and time with regard to their results in previous problems.

For GA, the differences in the parameters made a significant effect on the results. It gave the best results when learning_rate=0.1 and mutation_prob=0.25, which shows that there are more to learn and with more iteration, better results could come.

Training:

	precision	recall	f1-score	support
0	0.72	0.82	0.77	1262
1	0.71	0.73	0.72	1610
2	0.81	0.64	0.72	1128
micro avg	0.74	0.74	0.74	4000
macro avg	0.75	0.73	0.74	4000
weighted avg	0.74	0.74	0.73	4000
samples avg	0.74	0.74	0.74	4000

Testing:

	precision	recall	f1-score	support
0	0.73	0.87	0.79	315
1	0.74	0.75	0.75	403
2	0.82	0.63	0.72	282
micro avg	0.76	0.76	0.76	1000
macro avg	0.76	0.75	0.75	1000
weighted avg	0.76	0.76	0.75	1000
samples avg	0.76	0.76	0.76	1000

Genetic Algorithm got 0.75 f1-score, which is significantly worse than what Simulated Annealing got. This might be caused by the super dependent nature off weights in a neural network. In this case, the combination of two different states can be not so effective like the example in Flip-flop problem.

When training and testing scores are compared, it can be seen that none of these algorithms can overfit the dataset. It is clear that GA also underfits.

5.4. Conclusion

For Simulated Annealing algorithm and Hill Climbing, different parameter didn't make up any considerable difference in the scores of neural networks. I think this is because the neural network has a lot of weights and so easy to stuck in the local optima even though we could overcome the low local optima using Simulated Annealing. After the temperature is down, none of them could overcome the relatively high local optima.

GA didn't work well because differences in the weights are so dependent to each other in a neural network and combining them isn't a great idea according to the experiment.

8. Genetic Algorithm

[1] J. Vanschoren, "wine," OpenML. [Online]. Available: https://www.openml.org/d/187. [Accessed: 10-Feb-2020].