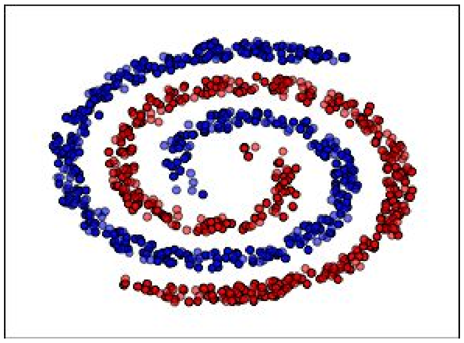
1 Neural Network Without Back Propagation

1.1 Two spirals



To test the neural network without propagation, I revisited the two spirals problem from assignment 1. In this dataset, the model is trying to predict whether a certain point is either a red dot or a blue dot. In assignment 1, I initially started with a low AUC of 0.62, but after tuning hyperparameters I was able to get an AUC of 0.96. In assignment 1 I tuned the hyperparameters and I got the best results with 20 layers with 100 nodes and I kept the same structure when testing with the new optimization algorithms.

1.2 Performance

Algorithm	Accuracy	Train Time
Back Propagation	0.98	0.70
Random Hill Climb	0.68	452.869
Simulated Annealing	0.50	453.154
Genetic Algorithm	0.66	24277.037

There are a couple interesting results from the table. First is that back propagation outperformed the other three algorithms significantly. In addition, the training time for back propagation was also the fastest by a huge margin. It’s not a completely fair comparison though, because the back propagation time I used is from Scikit-learn’s implementation of MLP. I’m guessing their implementation is highly optimized so a better comparison would have been to use back propagation using ABAGAIL. Going back to the accuracy performance, I was surprised at how much better back propagation was than all of the other choices. To gain some insight, I plotted the error results for each algorithm

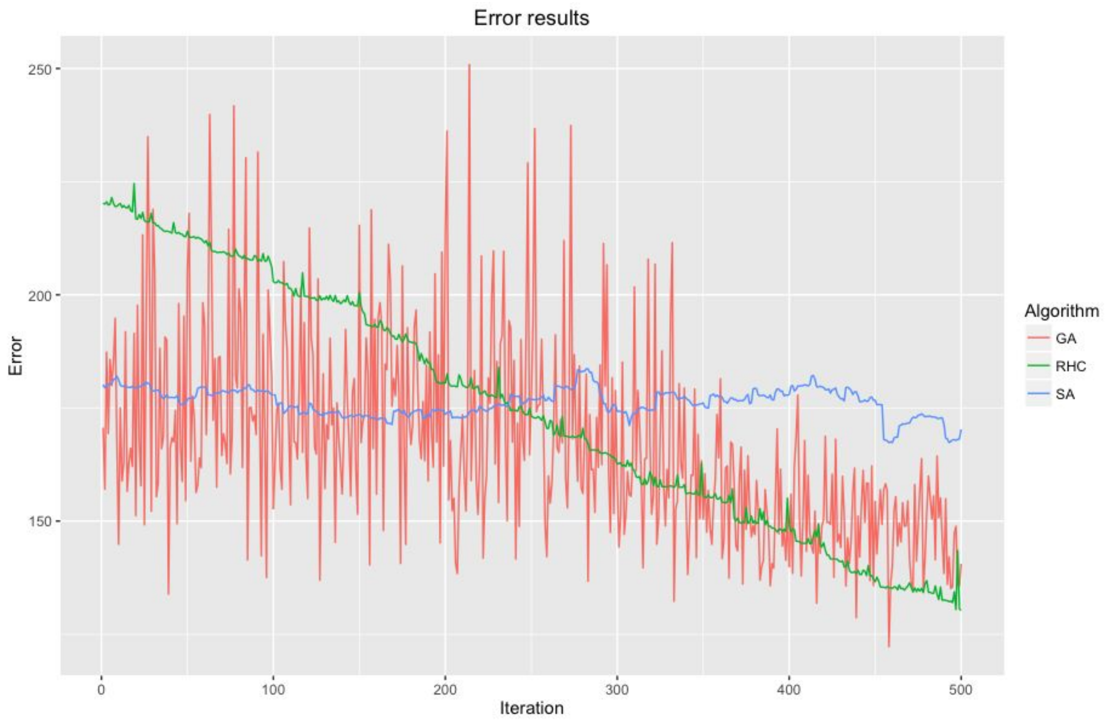


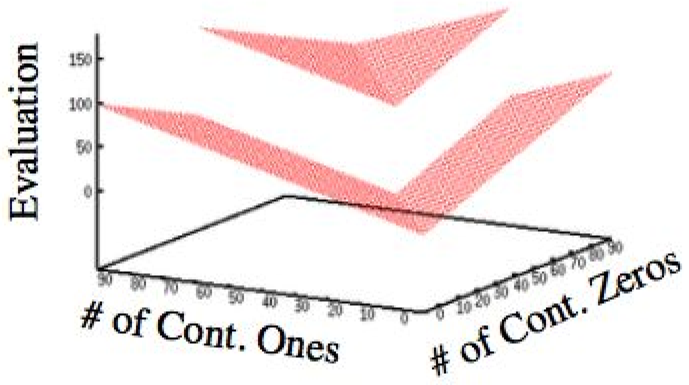
Figure 1: Error for each algorithm over number of iterations

The graph was interesting to me because it showed me that I am probably not allowing enough iterations for RHC and GA to find the optimal solution. RHC error is decreasing at a linear rate, and GA seems to be converging towards the end. The plot also shows me that SA's error remains constant and doesn't improve over iterations.

1.3 Conclusions

After testing and comparing the four optimization algorithms, to me it is clear that back propagation works best for the two spirals problem. There are still some unanswered questions because I did not test RHC and GA with a higher number of iterations, but because back propagation results are already close to perfect and the amount of time it takes to run the RHC and GA algorithms I think it's safe to pick the winner.

2. Four peaks



Four peaks is a fitness function that is interesting because there are two global maxima and two suboptimal local optima. The function is defined as follows:

$$z(x) = \text{Number of contiguous Zeros ending in Position 100}$$
$$o(x) = \text{Number of contiguous Ones starting in Position 1}$$

$$REWARD = \begin{cases} 100 & \text{if } o(x) > T \quad z(x) > T \\ 0 & \text{else} \end{cases}$$

$$f(x) = MAX(o(x), z(x)) + REWARD$$

Using this definition, the global maxima is 2N - T - 1

2.1 Optimization performance

To test performance of each optimization algorithm, I ran each one 20 times and averaged the optimum value reached. I also capped the number of iterations at 10000 in the interest of time.

After tuning parameters, the best parameter for SA was a cooling rate of 0.6. For GA, the optimal parameters were a population size of 90, mating size of 25, and mutation rate of 5. For MIMIC, the best performing parameters were a sample size of 200 and keeping 5 samples.

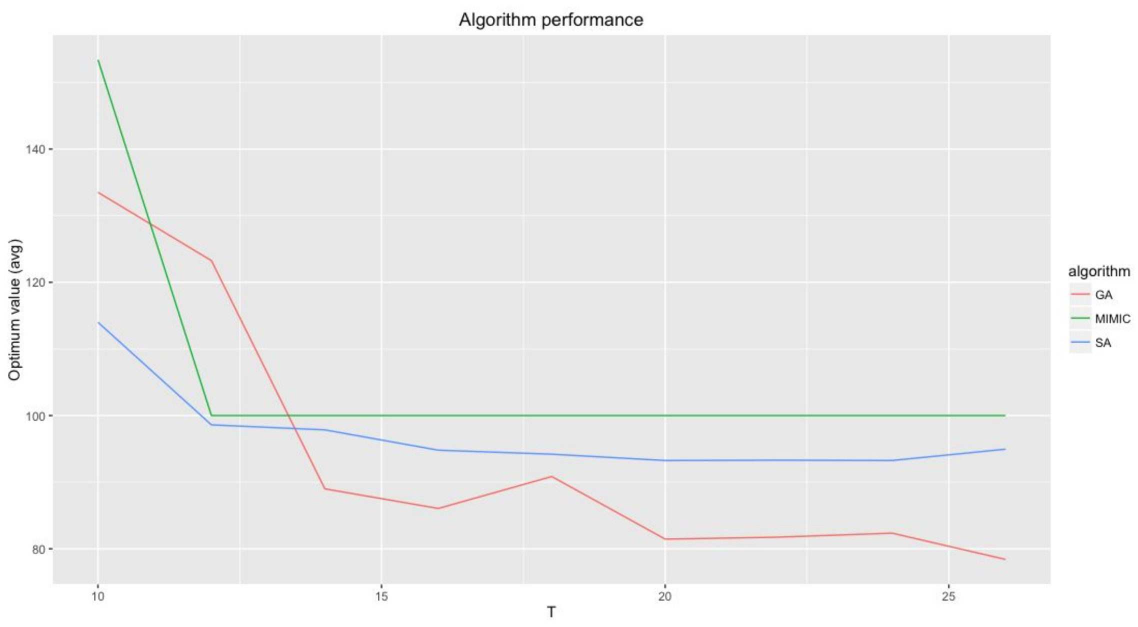


Figure 2: Performance of the three algorithms. X axis is the T value and Y axis is the average optimum value for each algorithm at that T.

As seen in the graph above, MIMIC consistently outperformed both GA and SA. Interestingly, as T increased the MIMIC performance leveled off while GA's performance kept decreasing.

2.2 Time performance

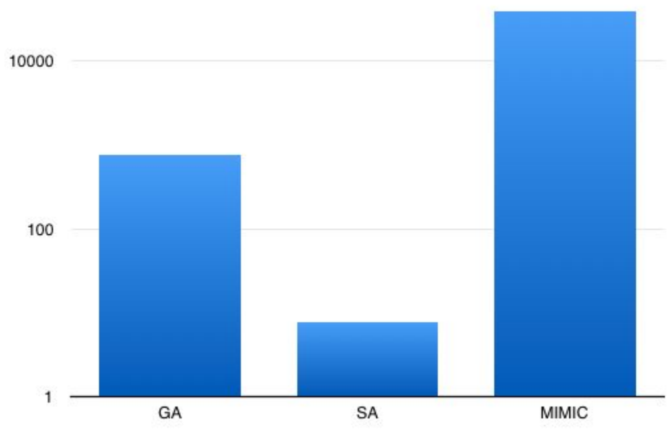


Figure 3: Wall clock time for the three algorithms. Y axis is measured on logarithmic scale.

Looking at the time performance of each algorithm, there were significant differences. SA performed the fastest, and MIMIC was the slowest by a large margin.

2.3 Conclusions

Taking into account the optimization performance and time it took to perform, picking the best optimization algorithm for this problem becomes difficult. MIMIC performed the best in finding the optimum value, but was orders of magnitude worse in performance time. If I had to decide on which algorithm to go with, if my T value was below 12 I would use MIMIC because of it's superior job in finding the optimum value (153.4 compared to 114.0). However, any T value 12 or above I would choose SA. The optimum value it found was on average only 6% less than MIMIC, but it's wall clock time was over 6000 times worse. This also of course depends on the application, as a 6% better solution would be worth the extra wall time if we were, for example, predicting heart failure.

3 Knapsack Problem

The knapsack problem is a combinatorial optimization problem that is NP-hard. Given a certain set of items, each of which has a weight and value, calculate the maximum value that you can hold in your knapsack.

3.1 Optimization performance

For this problem, I ran the three algorithms over various numbers of items that can go in the knapsack. I capped the number of iterations at 5000, and ran each algorithm 20 times and averaged their optimum value to measure the performance.

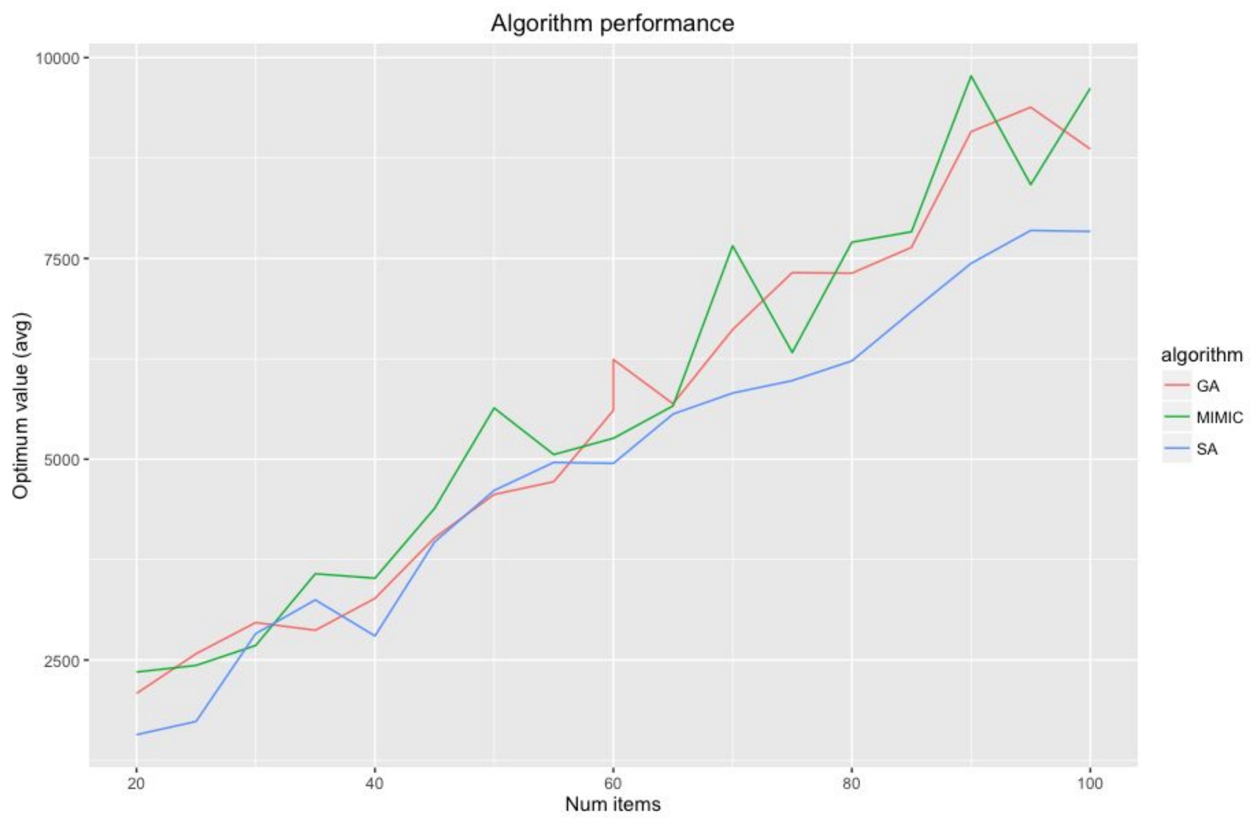


Figure 4: Performance of the three algorithms. X axis is the number of items for the knapsack, and the Y axis is the average optimum value found.

As seen in the graph, all three algorithms were pretty close but MIMIC had the top value in 11 of the 18 trials. It is worth noting that SA seemed to have more separation than the other two algorithms as the number of items increased. It's also interesting to split the data into two windows, one where number of items below 60 and above 60. When only looking at performance when number of items is below 60, GA only achieves the highest score on two of the 9 observations. When looking at 60 and above, GA achieves the highest score on 4 out of the 9 observations and never scores below SA.

3.3 Time Performance

The time performance of each algorithm is pretty much identical to the previous problem:

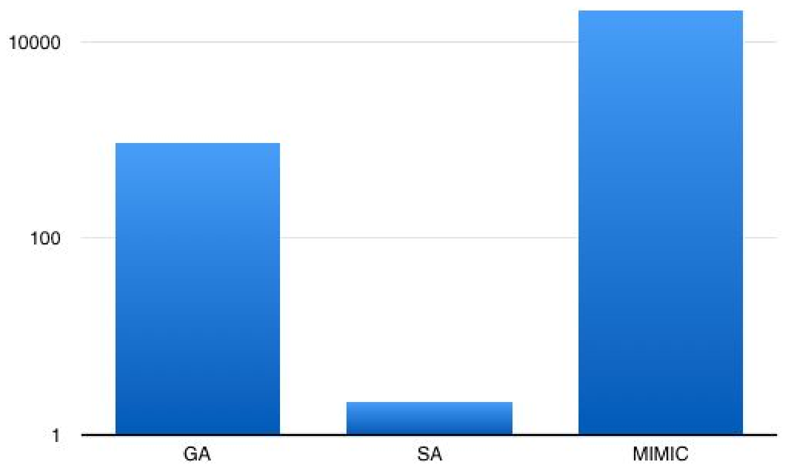


Figure 5: Time performance of the three algorithms. Y axis is measured on logarithmic scale.

MIMIC continues to be the slowest, followed by GA, and SA is significantly faster than both. It's worth noting again that for each algorithm I capped the number of iterations so it makes sense that this graph is so similar to the one in the Four Peaks section. A better way to measure time performance would be to measure how many iterations it takes to reach the global maximum (on average) and then measure how long each iteration takes.

3.4 Conclusions

MIMIC and GA outperformed SA in terms of reaching the global maximum, but SA is the fastest algorithm. For lower item values, SA had optimum values that were very close to MIMIC and GA and sometimes outperformed GA, so for a low number of items I would say that SA is the best algorithm. For higher number of items, SA didn't do as well and MIMIC and GA were neck and neck in terms of finding the optimum value. Because GA has faster performance than MIMIC, I think GA is the best algorithm for higher number of items.

4 Count Ones

Count ones is a simple problem where the objective is to get the counter of variables that equal 1. In this problem the optimal solution is N. Count ones is interesting in this case because it is an exhaustive search, whereas the other problems had a more complex nature.

4.1 Optimization Performance

I ran each algorithm over a range of N values, ranging from 60 to 200 and ran each algorithm 20 times and average their optimal value. Plotted below is each algorithm's average optimum value subtracted from N, so in this graph a higher difference means worse performance

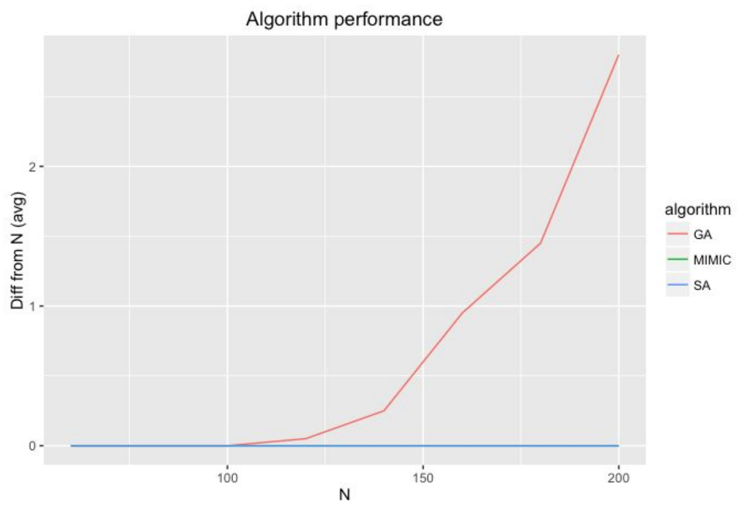


Figure 6: Algorithm performance for each value of N. Y axis is the difference between the optimum value and the optimum reached by the algorithm.

MIMIC and SA found the global maximum for every value of N, but as N increased GA got increasingly worse.

4.2 Time performance

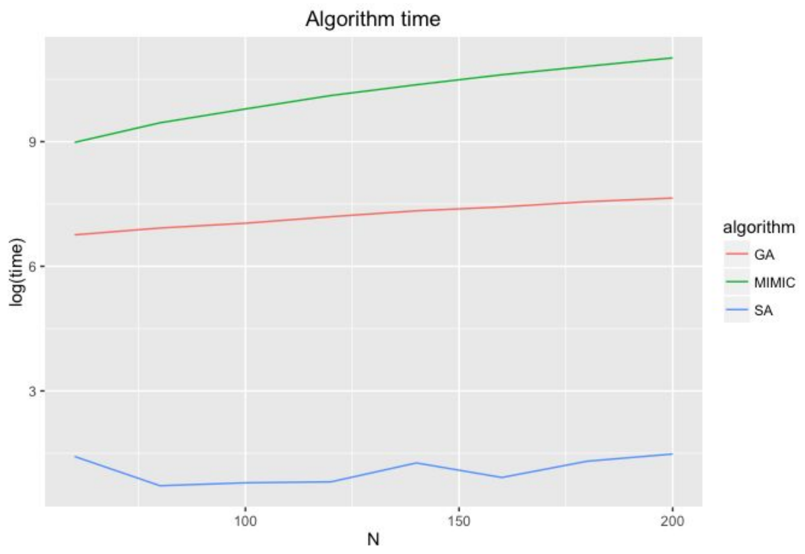


Figure 3: Wall clock time for the three algorithms. Y axis is measured on logarithmic scale.

In the graph above, each algorithm's clock time was plotted for every value of N. As seen in the previous problems, MIMIC was the slowest, followed by GA and SA.

4.3 Conclusions

In the count ones problem, there is a clear best algorithm. SA was able to find the optimum value for every value of N and had the fastest performance by a wide margin. MIMIC was also able to find the optimum value for every N, but is much slower and GA did not find the optimum value as N increased.

5 Conclusions

Applying the different optimization algorithms to the different problems has shown some patterns and takeaways. MIMIC tends to be the best at finding the optimum value, but is also the slowest. GA is faster than MIMIC but did not perform as well in terms of finding the optimum value, except for certain scenarios in the knapsack problem. SA generally had similar performance to GA in finding the optimum value, but is significantly faster. In the case of the count ones problem, SA was able to find the optimum value every time and also had the fastest performance.

These problems show that which algorithm is best depends on the type of problem and also the time constraints. SA very fast and does a decent job of finding the optimum value, and sometimes a very good job depending on the nature of the problem. MIMIC overall was the best at finding the optimum value but was computationally expensive.