Randomized Optimization

Zhijian Li

1 Dataset

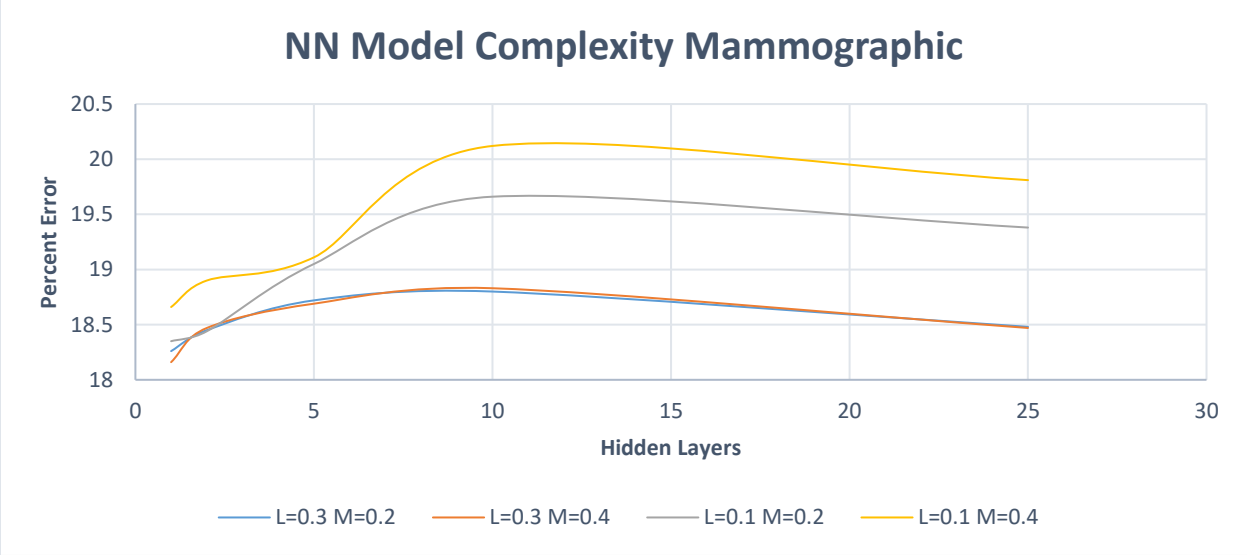
I will be using the Mammographic Mass Data Set from Project 1. For a recap:

Mammographic Mass Data Set (MM dataset): This dataset contains data from mammograms. The goal is to predict whether the patient has malignant breast cancer. The features contain BI-RADS assessments along with related measurements of the patient. The dataset contains 961 instances and 6 attributes.

2 Comparison

By using randomized optimization to find the best weights for the neural net, I will have a shorter training time, and depending on the optimization algorithm, I will have searched through a much bigger space than backpropagation and more likely than not, train a much more accurate neural net. Also, as you will see for many of the algorithms, because my dataset does not consist of too much data and there are only 6 attributes to worry about, with more iterations of each optimization algorithm, the accuracy improves dramatically, possibly because it has searched through almost the entire sample space.

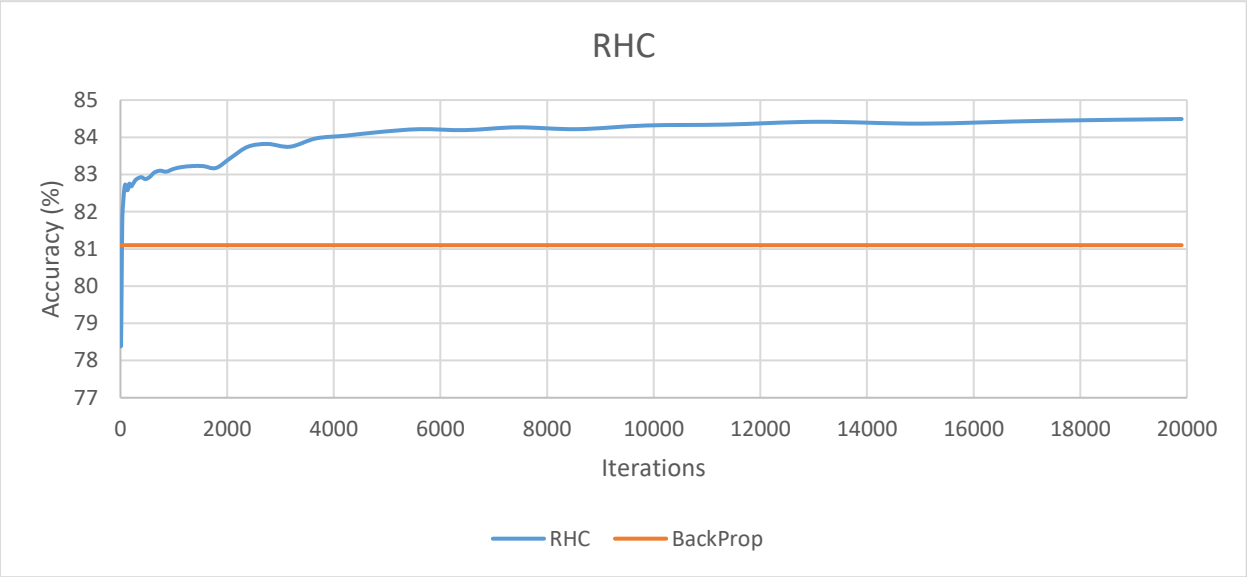
I will be comparing three randomized optimization algorithms for weight optimization of a neural net with backpropagation for the neural net in project 1. I will be analyzing Randomized Hill Climbing (RHC), Simulated Annealing (SA) and Genetic Algorithms (GA).



I will use the hyperparameters found in project 1 for neural nets, with 1000 epochs, a learning rate of 0.3, momentum of 0.2 and 2 hidden layers. With these parameters, the best accuracy for this dataset with neural nets was 81.1% accuracy.

Randomized Hill Climbing

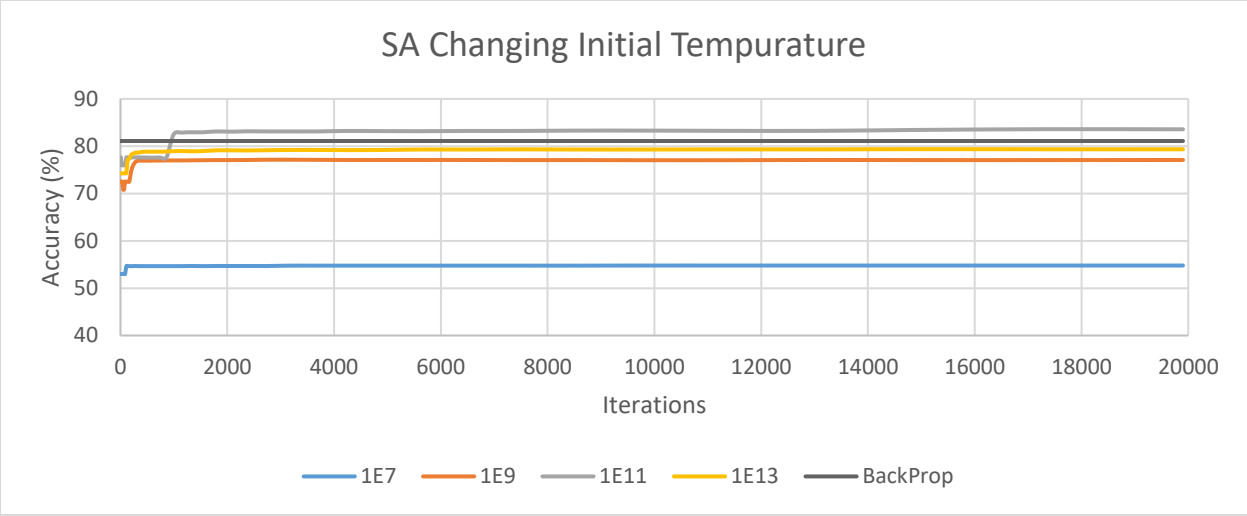
RHC was run with 5 trials, 20,000 iterations each. The graph plots iterations on the x-axis and accuracy along the y-axis. The final accuracy of RHC after 20,000 iterations was 84.473%. To speed up training times, I did not loop through every iteration, from 1 to 20,000. Instead, My iterator went from 1 to 20,000 and at each step, the iterator was scaled by 1.15 and shifted by 10.



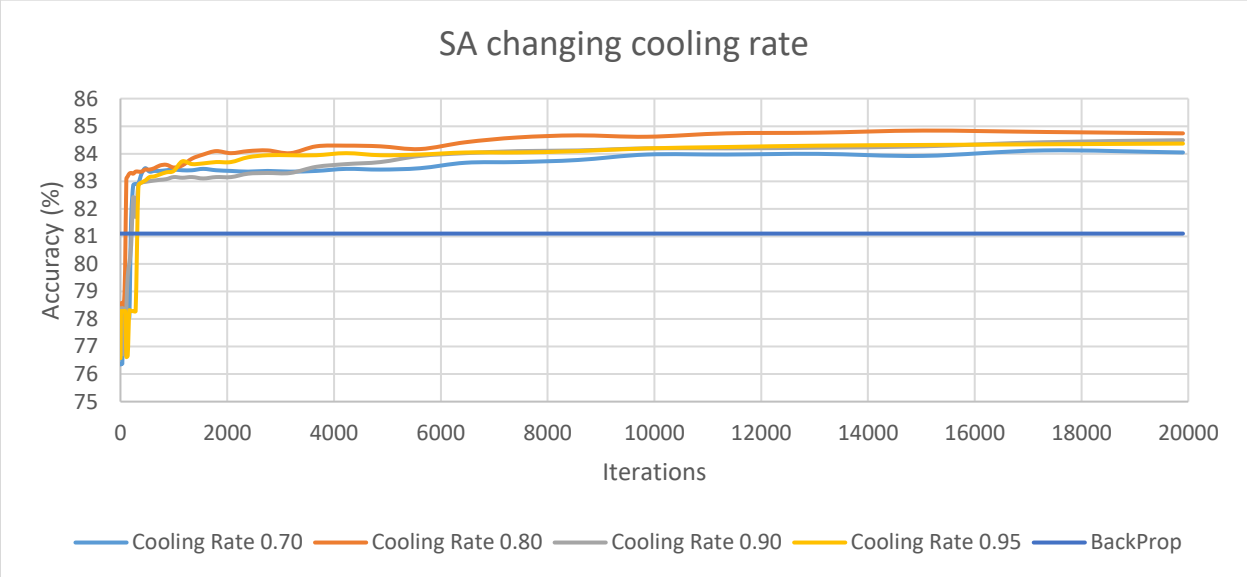
From the graph, accuracy steadily improves very close to the beginning. Even with 5 trials, the beginning has high variance in accuracy. After the initial 100 to 1000 iterations, RHC's accuracy already overcomes backpropagation. As RHC is ran with more iterations, accuracy continues to increase. One possible explanation is the search space is not that big, with 961 instances and only 6 attributes. Despite having no random restarts, RHC converges to a similar accuracy each trial. This suggests that the local minima were easy to find. One possibility is that the search space contained many local minima so the chance of randomly picking a point and having it be the local minima was high. Another possibility was that there were only one local minima and there was no way for the algorithm to get stuck at any other solutions other than that one.

Simulated Annealing

SA was also run with 5 trials and 20,000 iterations each. The graph plots iterations on the x-axis and accuracy along the y-axis. The final accuracy of SA was achieved with a cooling rate of 0.95 and an initial temperature of 1E11.



Temperature	Accuracy	Training Time
1.00E+07	54.79	33.1964
1.00E+09	77.10	32.1298
1.00E+11	83.54	33.7592
1.00E+13	79.36	32.2822



Cooling Rate	Accuracy	Training Time
0.7	84.0444	39.8732
0.8	84.7392	36.3828
0.9	84.4912	34.3748
0.95	84.367	31.1446

Running SA for 20,000 shows that SA converges to a set accuracy and maintains that accuracy after a mere 1000 to 2000 iterations. Two experiments were conducted, one that varied temperature and fixed cooling rate while another fixed temperature and changed cooling rate.

Varying Temperature

Cooling rate was fixed to be 0.90. With lower temperature, the algorithm performed very poorly. With a starting temperature of 1E7, the best accuracy achieved, even with 20,000 iterations and 5 trials was only an accuracy of 54.79%. Also, with a temperature of 1E13, while it did not perform as bad, got an accuracy of 79.36%. This suggests that the optimal temperature is 1E11. One explanation of this is that if temperature is too low, the algorithm is not given enough chances to explore the entire search space and that there are only few local minima, contrary to the conclusion from RHA. Not given a higher enough temperature is the same as having few random restarts. With a search space that contains only a few local optima, the chances of landing on a local optimum randomly, or finding one in your neighborhood is very low. This can attribute to the very low accuracy with a starting temperature of 1E7.

With a starting temperature of 1E13, the algorithm also did not perform optimal, as expected due to the limited number of local optima in the search space. One explanation is that the temperature was so high, the algorithm was not given enough chances to further explore potential local optima, but jumped too often initially, landing on a less optimal solution and sticking to that.

All variations on starting temperatures did not outperform neural nets, except a starting temperature of 1E11. At this temperature, the algorithm outperformed NN around 1000 iterations. This just shows how much better SA performs compared to backpropagation. At 1000 iterations, it only took roughly 2 seconds to find the neural net weights, while backpropagation at 1000 epochs took roughly 1 minute to train.

Varying Cooling Rate

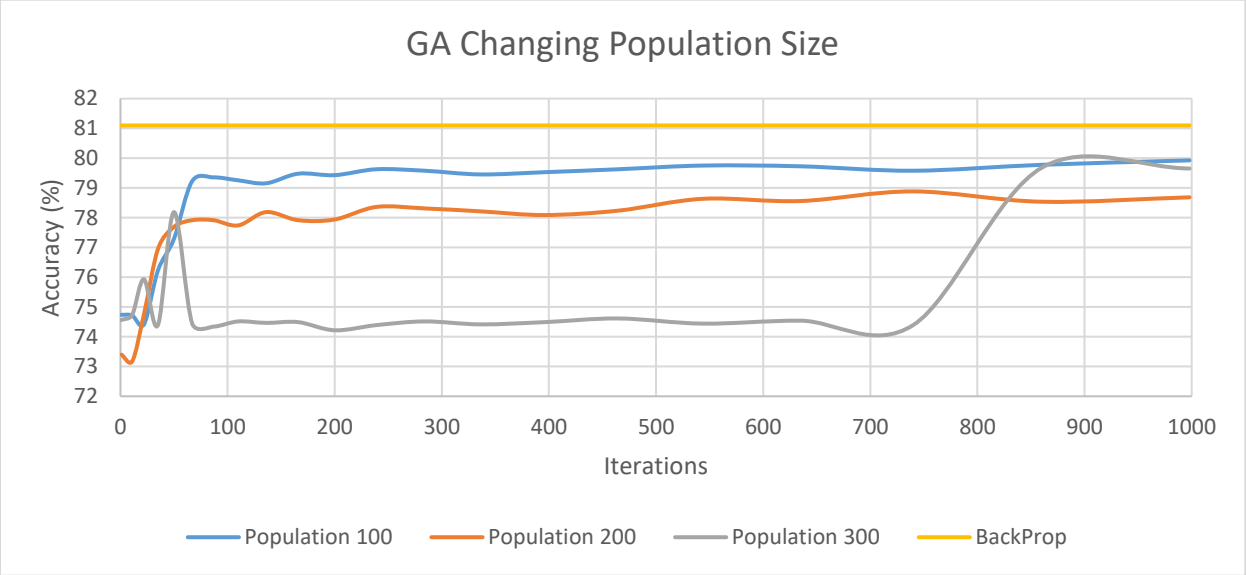
With the best temperature being 1E11, that was the fixed starting temperature. When temperature was fixed and cooling rate was modified, temperature changes affected accuracy much more than cooling rate. It might also be the case that a wider range of cooling rates should have been tested, despite cooling rates of 0.7 and 0.95 being vastly different. The cooling rate that produced the best accuracy was a cooling rate of 0.80.

All cooling rates outperformed backpropagation, and this time it only took about 300 iterations for SA to surpass backpropagation. After that many iterations, SA converged and maintained accuracy. This makes sense because as iterations increase, the results should be smoother. With SA outperforming Backprop at around 300 iterations, it takes not even half a second to find optimal weights.

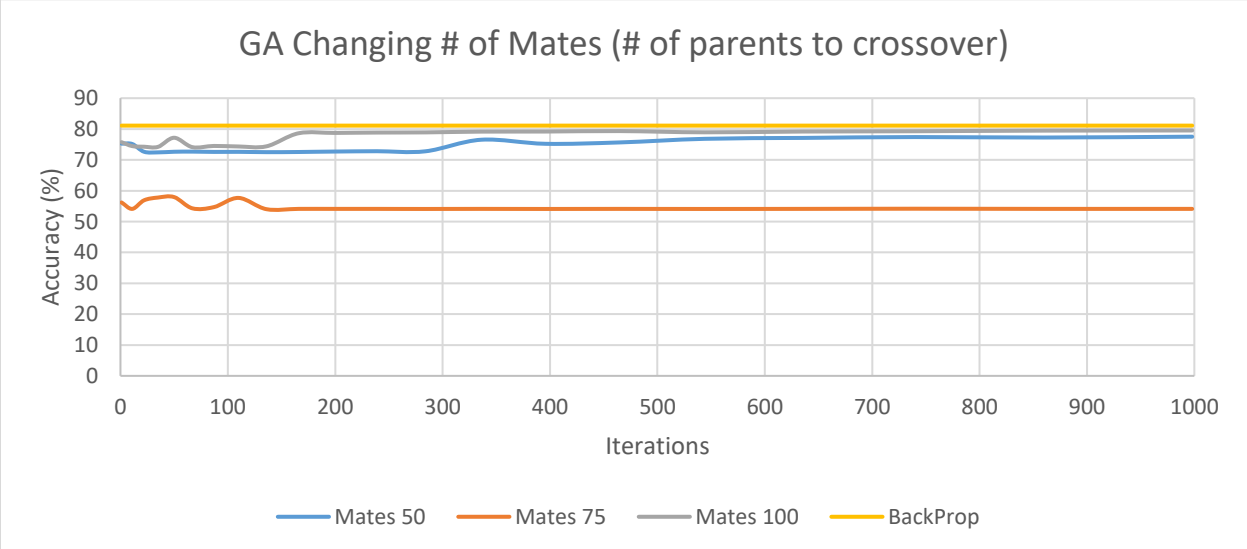
An interesting observation is that with a cooling rate of 0.70, the algorithm took the longest to run, which is counterintuitive as lower cooling rate means temperature decreases slowly, because at each step, temperature is set to be temperature * cooling rate.

Genetic Algorithm (GA)

Genetic Algorithms were run with different parameters. Instead of 20,000 iterations, GA only ran for 1,000 iterations. This is because, unlike RHC and SA, GA takes significantly longer to run, mainly because it doesn't not simply find a single best weight, but finds a sample or population of good weights. Even with much fewer iterations, GA still converges to one accuracy. Two parameters I changed were the population size per iteration and the number of mates per iteration (the number of crossovers essentially).



Population Size	Accuracy	Training Time
100	79.9254	24.5018
200	78.6848	22.9856
300	79.6528	37.8502



# of Mates	Accuracy	Training Time
50	77.5188	40.7424
75	54.1188	27.8784
100	79.5286	31.554

Overall, GA did not perform nearly as well as backprop and could only match backprop at best.

Changing Population Size

Population size was varied between 100, 200 and 300 while mates and mutations were both fixed at 100 and 10 respectively. With a population size of 100 and 200, GA quickly converged to an accuracy and stuck with it (between 79% and 80% accuracy). With a population of 300, it took much longer to converge to a good accuracy (800 iterations). This makes sense as a bigger population has a higher chance of incorporating bad weights (weights that give lower accuracy), especially if the dataset contains few local minima as seen in SA and RHC.

Despite varying populations, GA does not seem to perform as well as BackProp, RHC and SA. Even as population size increased, accuracy remained the same, but that does not mean that there is no good population size that will produce a good accuracy. But, GA still runs faster that backprop.

Changing the Number of Mates per iteration

Population size was fixed to 200 and the number of mates were varied from 50, 75 and 100. Unlike changing population size, there was much less fluctuation in accuracy as seen in the graph. But it is also very clear that having 75 mates had a much lower accuracy. This is interesting, as it is saying that having more mates or having less mates is better as opposed to having the number of mates be in between.

The performance of GA was much closer to Backprop when the number of mates were changed. The best accuracy or 79.53% was achieved with 100 mates. This suggests that GA is not the best algorithm because unlike RHC and SA, the samples generated with GA is much less random. It does not randomly search through the sample space, but from one randomly generated sample space, tries to evolve a good sample space through mutations and crossovers of the initial sample space. This limits GA in that it does not get to explore the sample space enough, because it only completely randomly generates a sample space in the beginning. Essentially, GA does not have a random restart mechanic that SA has, and it is restricted to a sample space once initialized, unlike RHC, where it jumps and randomly searches the space.

Another explanation could be that GA simply did not run enough iterations. Also, different mutation rates were not explored, which could have led to better accuracies. More specifically, if there were more mutations, GA would be more random and more like SA and RHC, which might allow it to produce better accuracies.

3 Conclusion

In conclusion, randomized optimization is clearly a much better algorithm for weight tuning that backpropagation for the Mammographic dataset. Both RHC and SA clearly outperformed Backpropagation, both in accuracy and training time. This makes a lot of sense, as this dataset was small and contained only a few attributes. This presented a small enough search space that RHC and SA found local minima easily, due to fast training times which led to more iterations. Despite this fact, GA could not perform as well.

This suggests that despite the small dataset, it was still complex enough that GA performed worse. This is further evidenced by the fact that no algorithm thus far has produced an accuracy that I would say is good enough, with the best accuracy being only around 85%. A dataset that seems simple based off its size and attributes still maintains complexity that cannot be captured by any algorithm so far.

4 Optimization Problems

The 3 optimization problems I chose were FlipFlop, Traveling Salesman and Four Peaks. Knapsack highlights SA's strengths while TravelingSalesman highlights GA's strengths and FourPeaks highlights MIMIC's strengths.

FlipFlop

The FlipFlop test is a problem that counts the number of alternating bits in a bitstring. The optimal solution is when the bitstring consists of only alternating 0's and 1's. The tests were done with bitstrings length 100.

Algorithm	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
RHC	78	84	76	82	82	80.4
SA	99	98	99	99	99	98.8
GA	92	90	85	88	85	88
MIMIC	93	94	95	95	94	94.2

SA performs the best because the problem contains many local minima. This allows for SA to perform well because it can initially search through the space and eventually settle in a local optimum, which will eventually lead to a global optimum. In the specific case, SA is checking the neighborhood consisting of a single bit flip. As bits are flipped, eventually it will lead to the global optima, as it consists of sequential bit flips. SA works well because it does not get stuck in bad local optima, since there are so many and with a high enough temperature, it will be able to escape.

GA does not perform as well because it must slowly work its way towards a solution. Also, unlike RHC and SA, it does not have a random restart mechanic, so when it gets stuck a local optimum, it cannot escape it.

MIMIC also does not perform well because if there are a lot of local optima, the probabilities will be very similar, and the distribution will be more flat instead of more closely like a

Gaussian. That is, there are so many solutions, MIMIC will be able to find a solution, but it cannot work its way towards the best solution.

In conclusion, SA works the best because it locates various local optima, with the ability to escape it, while GA easily gets stuck in these local optima and MIMIC has to search through every solution, many of which are not the best.

TravelingSalesman

The TravelingSalesman problem is given a graph, finding a shortest path that visits every node. The graph contains N vertices, and the weights for each edge is randomly generated. The tests are done with N = 50. The specific test doesn't minimize the length of the path, but instead tries to maximize the inverse of the length of the path.

Algorithm	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
RHC	0.138	0.12	0.134	0.112	0.111	0.123
SA	0.116	0.119	0.124	0.125	0.118	0.1204
GA	0.148	0.166	0.164	0.146	0.134	0.1516
MIMIC	0.092	0.093	0.095	0.091	0.087	0.0916

GA performs the best for this test. This can be explained by how the problem is set up. Because the TravelingSalesman problem is a graph search, the solution is NP complete. Because it is a graph, the search space is incredibly huge, at every node the search space increases exponentially, that is there is an exponential amount of possible solutions to search through. Both RHC and SA perform poorly because they search randomly in the search space. Because the search space is so big, it is very unlikely that they will find an optimal solution by randomly searching.

MIMIC tries to find the probability that good solutions exist in the current search space. This will lead to an optimal solution, but it will take extremely long because it must consider all possible solutions in the search space. It is still better than RHA and SA because it does not blindly search through the search space, it takes possible solutions and elevates them.

GA works very well because it takes a given sample space, and only further *evolves* possible solutions that are more likely. That is, it takes good solutions and tries to find better ones by making small changes. This works because for any possible solution, making small changes in the right direction will lead to a local optimum. Unlike MIMIC, it does not need to search through the entire search space, but instead takes a small sample space and grows it until it contains a good solution. This takes much less time as it only needs to care about its current sample space.

Four Peaks

The Four Peaks problem can be described as counting the max number of contiguous 0's or 1's in a bitstring length N up to a certain point T, and then counts the number of the opposite bit in

the 2nd half. More specifically, it counts the number of leading 1's and trailing 0's. The function it attempts to maximize is the sum of the max of the # of leading 1's or trailing 0's and whether there are at least T trailing 0's and leading 1's.

It is called four peaks because it contains 4 optimal solutions, 2 global optima and 2 local optima. The 2 global optima are when the bitstring consists of T + 1 leading 1's and the rest are 0's or T + 1 trailing 0's and preceding 1's. This gives us (N – T – 1) +N or 2N-T-1 as an optimal solution. The suboptimal solutions are when the bitstring consists of all 1's or 0's, in which case the solution is N.

Another note is that the optimal points are inside peaks, where those peaks are very narrow.

The test was done with N = 100, T = 10 with the optimal solutions being 100 and 159.

Algorithm	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
RHC	18	21	20	14	16	17.8
SA	8	3	13	8	8	8
GA	34	116	122	117	123	102.4
MIMIC	145	176	153	159	149	156.4

MIMIC outperformed all the other algorithms. This is because of how the problem is set up. While the search space is not as big as the TravelingSalesman problem, there are only 4 optimal solutions out of all possible bitstrings (2^N).

RHC and SA performed abysmally, having a solution not even 20% of the optimal. This makes sense because both RHC and SA must randomly jump through the search space or 2^N. Beaks the optima are in narrow peaks, the chances of randomly stumbling on one of those peaks in the entire search space is very low.

Similarly, GA, while it does not perform as poorly, still does not outperform MIMIC. This can be analogous to a man stumbling in the desert trying to find a water, starting in a random location and having an idea of where to go. The problem here is that, while given enough time, he will find the water source, the amount of time needed is too much as GA only takes a random location and tries to slowly walk towards that water source. More concretely, given a search space, GA will have some idea where to move to find the local optima, but takes far too long to reach it. (Funny enough, the 2 local optima can be analogous to desert mirages).

With MIMIC, it can be thought of as looking at the entire search space. Essentially, with the analogy, you are looking down at the desert and it is much easier to see where the water sources are. MIMIC takes the search size and deals with probability distributions pertaining to the space. The optima will have their probability of being found exponentially increase with each iteration.

In Conclusion, under similar parameters, MIMIC outperforms all other algorithms in the four peaks problem. Obviously, if RHC and SA were run with 20,000 iterations, then they would have a much closer solution, but that is no longer comparing apples to apples.

5 Conclusion

RHC and SA tend to work well when the sample space is big, but there are just enough optima that it is likely they will stumble upon one given enough time, and those optima are not far from the global optima. They perform poorly when the solution is something that can be easily worked towards, but there is a very low chance of randomly stumbling upon the answer.

GA works well when there is a clear solution to work towards and GA has enough time to work towards that solution. It must also be a problem where local optima are close to the solution, otherwise GA will be stuck at a bad optimum.

MIMIC works well in search spaces that are big, and contain only a few solutions. As seen in the FourPeaks problem, MIMIC performed well because, despite only having 4 solutions and the rest of the solutions are nowhere near the global optima, MIMIC can look at the entire space and the probability of those solutions are very high after a few iterations, while the bad solutions get washed out. MIMIC does poorly when there are so many solutions and global optima are sprinkled in there because the probability of the best solutions are still the highest, but not by much.

References:

1 https://archive.ics.uci.edu/ml/datasets/Mammographic+Mass

2 https://archive.ics.uci.edu/ml/datasets/Diabetic+Retinopathy+Debrecen+Data+Set

3 https://github.com/pushkar/ABAGAIL