ASSIGNMENT 2

Randomized Optimization

Amisha Buch (abuch6)

For this project we implement and analyse 4 different randomized optimization algorithms. Brief definitions of those algorithms are as follows:

1. **Random Hill Climbing:** RHC is an iterative search technique where the algorithm tries to find the global maxima in a search space by starting at a random position and moving in the direction of the steepest ascent. It works well for convex problems. (If required to find a global minima, it will move in the direction of steepest descent.)
2. **Simulated Annealing:** Simulated Annealing takes cue from the process of annealing in metal work, which involves heating and cooling a metal to alter its physical properties. SA does something similar, where the temperature will be initially set as “high”, which gives the algorithm the ability to jump out of any local optimums. As the temperature is gradually “cooled”, which allows the algorithm to focus on a search space where an optimum solution can be found.
3. **Genetic Algorithms:** Genetic algorithms work on principle of “evolution” and natural selection in various species. It finds a set of “good” points from the population, and then mutates them to find the points with the best qualities of both parents. The points with the bad qualities (which do not meet the expected threshold in terms of machine learning) are discarded. This is iteratively done, to find the optimal solution. Only disadvantage of GA is that it does not scale well with complex problems, I.e if the elements to mutate is very large in size.
4. **MIMIC:** Mutual-Information-Maximizing Input Clustering works on the principle of finding optimal by estimating probability densities. MIMIC is similar to neural nets as it has a “structure”, as opposed to other randomized optimization algorithms, and uses this structure for passing on information about the cost function from one iteration to another.

Lets start with applying these algorithms now and comparing them:

OPTIMIZATION PROBLEMS:

I have covered the Travelling Salesman, Knapsack and FlipFlop problems to compare and highlight the features of different optimization algorithms. All the code for the algorithms and problems is written in JPython (Python and ABAGAIL), and the plotting code is written in python. I have used the following references:

1. https://github.com/pushkar/ABAGAIL - ABAGAIL various optimization problems and how to implement them
2. https://github.com/JonathanTay - Got the idea of creating a error calculating function errorOnDataSet from this library.

PART 1

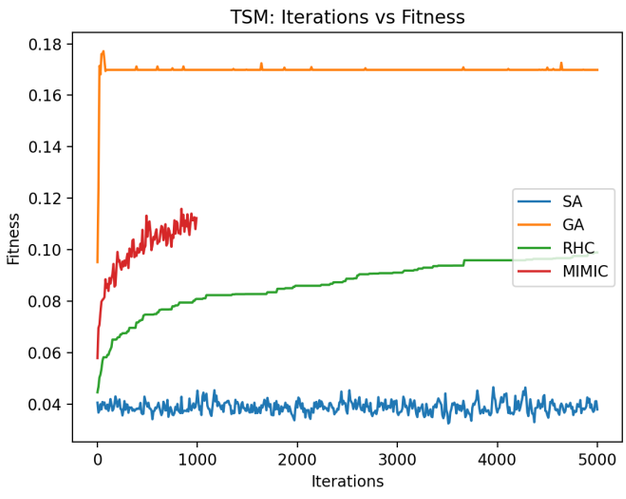
TRAVELLING SALES MAN PROBLEM - The travelling salesman problem (TSP) aims at finding the route with the shortest round trip between a set of cities, which are represented by 2D points. The condition here is that each city should be visited only once. This is a NP hard problem and is useful in many transport and logistic

companies, travelling companies as well as now in navigation services like Google maps etc. The direction of travel does not matter, given that we find the optimal route.

Parameters for the experiment:

- We consider 50 cities here, i.e N = 50
- I ran the GA, SA , RHC algorithm 5 times, with 5000 iterations for each run. We ran MIMIC for 5 times but with 1000. The results were averaged to compare and plot values.
- The distance metric here is 1/d, where d is the total distance between the start and end cities. Hence, the algorithm / solution which gives the smaller d, has the 1/d higher which is a better fit.

Let's see some graphs and compare the performance of the different algorithms.



Graph 1.1

Graph 1.2

As we can see from the graphs, Genetic algorithms reaches the maximum fitness value pretty fast. Simulated annealing performs the worst, with lowest fitness (accuracy). MIMIC may achieve similar fitness over longer runs, but as MIMIC consumes the most time (Graph 1.2), we still prefer GA here. Also, note that this is just for N = 50. Also, since this is a 2D problem, MIMIC needs to sort the samples to convert it into a vector, which also adds to the total time taken. As RHC and SA are similar in nature, they take similar times and performance is also similar compared to GA.

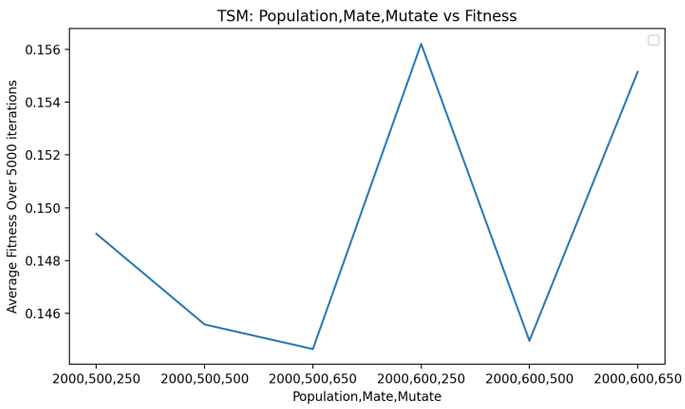
Why is GA best for this problem?

A route can be obtained by the permutation of various routes, while sorting the points(cities). As this process is similar to cross-over or mating, I.e finding the optimal combination of mates, GA works best for this problem. It is not like SA or RHC, where we take smaller steps in one direction to find optimal. The solution can even be optimised further by using swapping of cities (substituting neighbours method), but I have not tried that in the solution. Even though GA is a little volatile, it converges quickly in less than 100 iterations, other altos take a long time to get near optimal solution or converge. TSP problems have a large hypothesis space, but I think as it contains a lot of local optimal similar to the global optimal, which is why GA converges so quickly. As there can be many routes between cities, which nearly have the same distance to travel, but may not be optimal when taking the entire

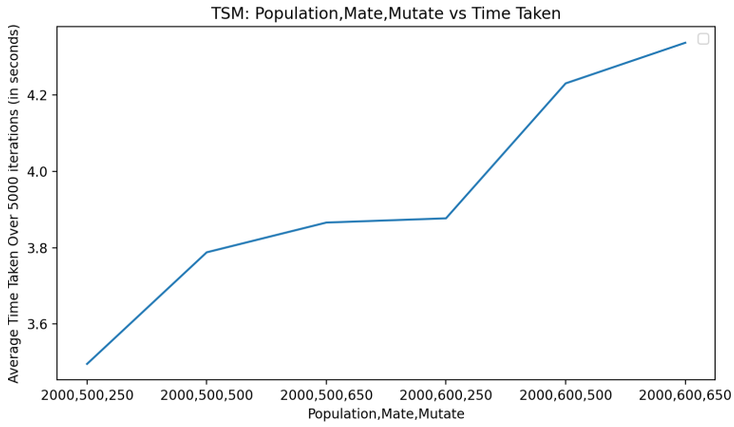
hypothesis. MIMIC performs considerably better than SA and RHC, but it works on probability densities of the data to find a structure. Here that does not matter as each city occurs only once in the solution, and the probability of it occurring does not affect the optimal route.

In order to find the best parameter, I tried a number of combinations for the population, mate and mutate variables. Population size represents the upper limit on the number of samples in the space at any given time (similar to survival of fittest). Mate represents the mating between samples and Mutate is the random changes in the samples to create better results.

The best results I found were at the combination (p, mate, mutate) -> (2000, 600, 250). Also, this result was obtained only within a 100 iterations. This combination also turned out to be the fastest to converge, which shows that the time taken is proportional to the Mutate value of the arguments. This can be seen in the graphs below.



Graph 1.3



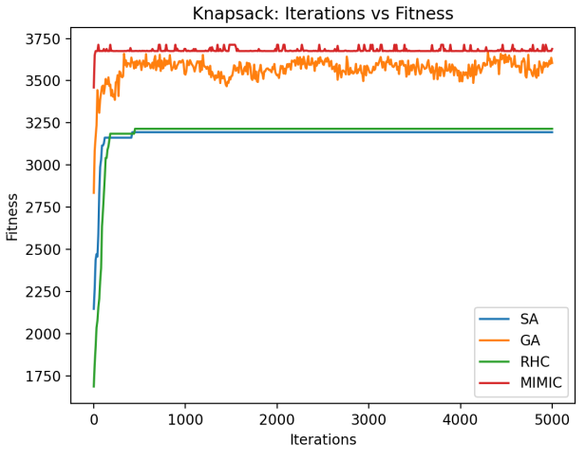
Graph 1.4

KNAPSACK PROBLEM

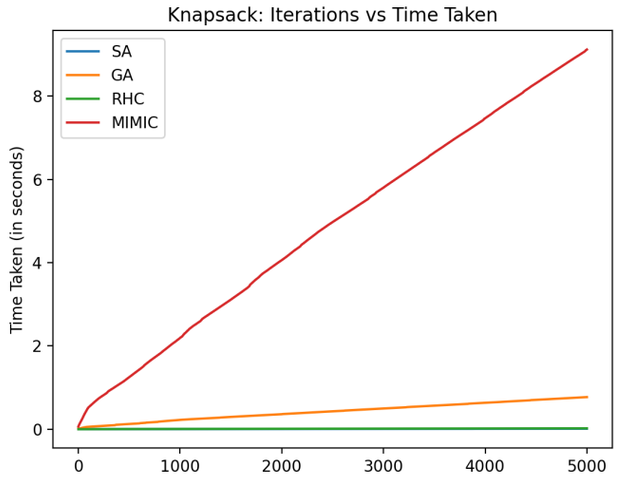
This is again an NP hard problem, which is a combination optimization problem. Given a number of items, each having a fixed weight and value, we need to find the quantity of each item to put in a knapsack such that the total weight is less than a given limit and the value of the knapsack is as large as possible. These kind of problems have many practical applications like maximising profit in financial markets, travel agencies for sporting equipment's etc. A simple example would be to take the most important items for a hike, in order to maximise efficiency.

Here, we fix number of items to 40, with 4 copies of each. The max weight and volume for a single item is. The maximum volume of the knapsack is set to max_volume * num_items * num_copies * 0.4 = 3200. We took 5 trials of each algorithm with 5000 iterations during each trial. The results are as follows:

MIMIC performs the best among all algorithms. GA comes to a very close second, and SA and RHC perform comparatively poorly. The fitness lines in the graph can even be smoother if we increase the number of iterations. The time taken by GA is significantly lower than MIMIC, but the accuracy is not as close to MIMIC that we can prefer GA over MIMIC. In the Knapsack problem the order in which the items are placed in the bag didn't matter. Building on the samples distribution might be the ideal way to approach this problem, which MIMIC does.



Graph 2.1



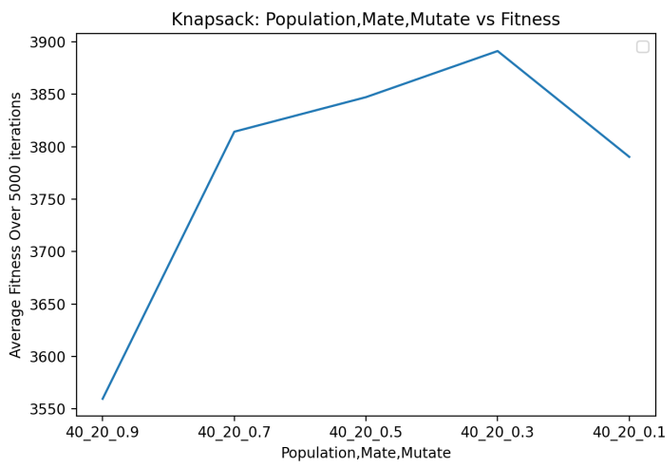
Graph 2.2

All the algorithms converged within less than 200 iterations. Still MIMIC takes the most time as it needs to calculate the mst in each iteration.

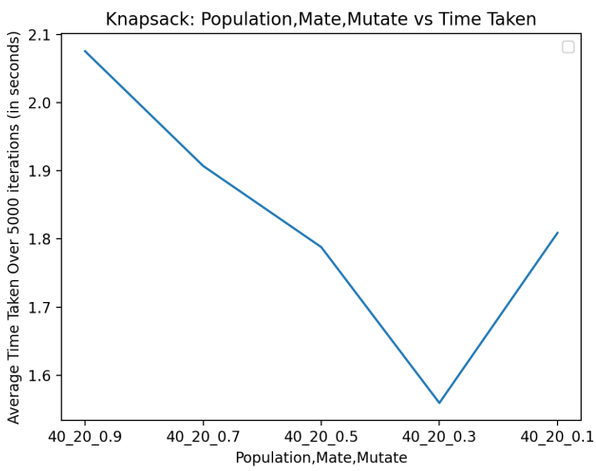
Why does MIMIC do better?

MIMIC performs well because it works on probability densities of the samples and represents the dependencies between them, by propagating the cost function in each iteration. This goes to show that it retains or remembers the “history” of the previous iteration. It uses this history and gain from it to get the best option in the next iteration. If GA and MIMIC perform almost the same, with GA taking much less time, why would we choose MIMIC? This is because as you see in Graph 2.1, the MIMIC plot is much more smoother, which Ga has some jitters, which shows that the variation in GA is much more, i.e the standard deviation must be higher for GA.

Next, we want to find the best parameters MIMIC. I tried various population sizes varying from 20-100 and to keep values from 10-50. The optimal values of the sample (population) and the toKeep values were 40 and 20, with 0.3 as the m value to the dependency tree. Here, “m” is the small positive value to add when making the tree.



Graph 2.3



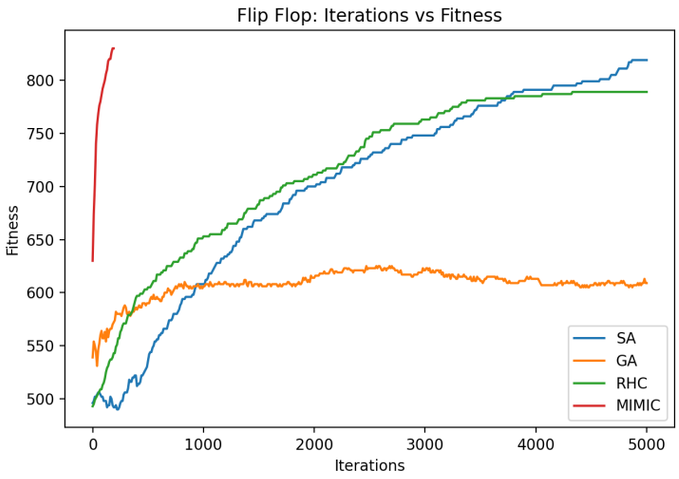
Graph 2.4

Also, the time taken for these parameters to converge was the least, with fitness value around 3900.

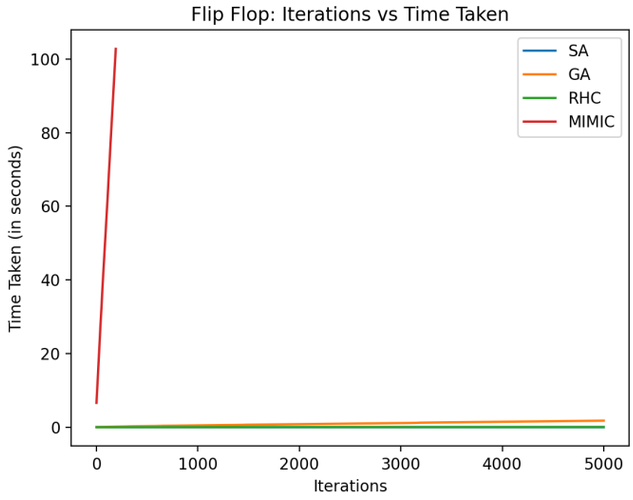
Note: Here, in graphs 2.1-2.4, please consider Samples,ToKeep,M instead of Population, Mate and Mutate.

FLIP FLOP PROBLEM

In the flipflop problem, we need to find out the count of bits that are different from it's neighbour in a binary string. The highest count would be obviously of the string which has alternating 0's and 1's. But this is the ideal scenario. The problem with flip flops is that there may be a long string of alternating strings in one part of the entire string and another longer string in another part. Hence, there are a lot of local maximal that would be present.



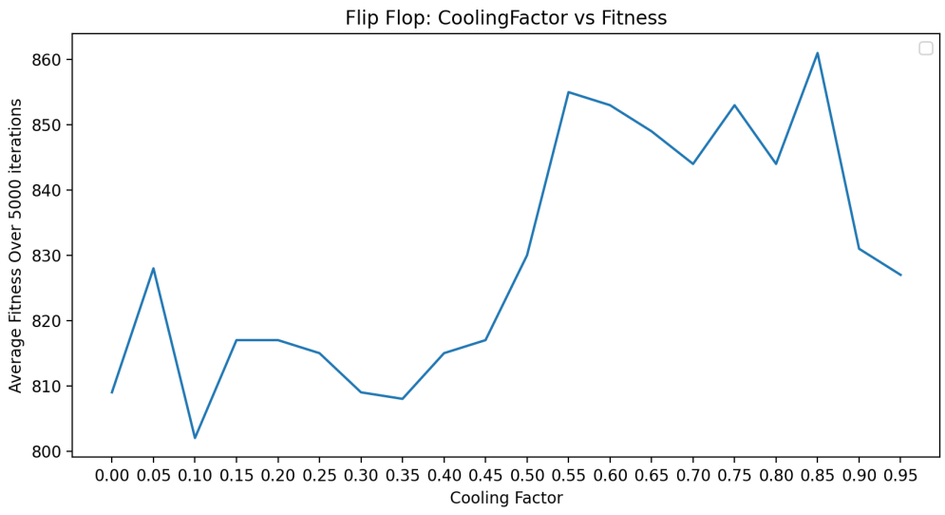
Graph 3.1

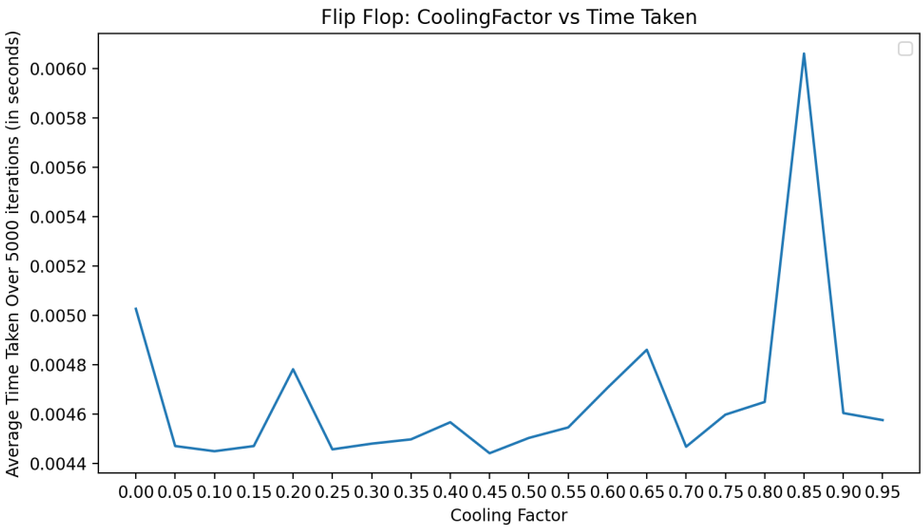


Graph 3.2

For the SA, GA and RHC, we ran a total of 5 trials with 5000 iterations each. The results were obtained by averaging over the 5 iterations. For MIMIC, we only ran 500 iterations for each trial, as it takes very high amount of time to converge. Here, we took N= 1000. Lets look at how all the algorithms performed:

As you can see from the graphs, GA performs the worst. Initially, what I saw was that RHC performs better than SA, but eventually as the iterations grew, SA performed much better and could find a better fit. MIMIC performs well as well and finds an optima much similar to SA, but the main issue with MIMIC is that it takes almost 1000 times the time taken to converge. SA takes time in the range of milliseconds, while





Graph 3.4

MIMIC took a minute on an average to converge. MIMIC still performs better in terms of no of iterations to converge, as it converges within 200 iterations, while others around 4500.

Why does SA do better?

SA is known to perform really well on a large search space. (Here N = 1000). It has a lot of local optimal which are close to each other, where SA works better than RHC. RHC should also perform well here, but the problem Is that once it finds a local maxima, it will randomly restart. Since we need to find an approximate global optima, rater than precise local maximal, RHC got a close value to SA, but is still lower. Maybe with larger iterations it will reach the fitness value of SA.

To get the optimal values with SA, I tried various values of the cooling exponent (CE).

From the graphs above, the optima was obtained when the cooling factor was around 0.85, which also takes the maximum time compared to other values. In order to even further optimise the results, we can maybe try reducing the number of bits - this may result in showing that SA is better than MIMIC in lower iterations.

We can also use different values of initial temperature rather than just changing CE.

PART 2

NEURAL NETWORKS WITH OPTIMIZATION PROBLEMS

In this section we apply three algorithms - SA, GA and RHC to find the optimal weights for the neural network we proposed for one of the datasets we used in Assignment 1. The dataset I have used is the Diabetic Retinopathy (DR) Debrecen Data Set. This data set contains features extracted from the Messidor image set to predict whether an image contains signs of diabetic retinopathy or not. The data is complete with 19 attributes, out of which 18 are features and one is the Result (0,1) variable which determines if the image contains sign of DR. There are 1151 samples in total. There are no missing values. The attributes in the dataset were normalised using the StandarScaler pre-processing library in python.

I first split the data into training and testing sets, in the ratio 80/20. I used the abalone_test.py from the ABAGAIL for the back propagation code and algorithms.

Three trials of 5000 iterations each were executed in order to get the test/train accuracies, as well as the computation times.

Comparisons were then done based on performance accuracies on the test data and computation times. The results I obtained were quite surprising.

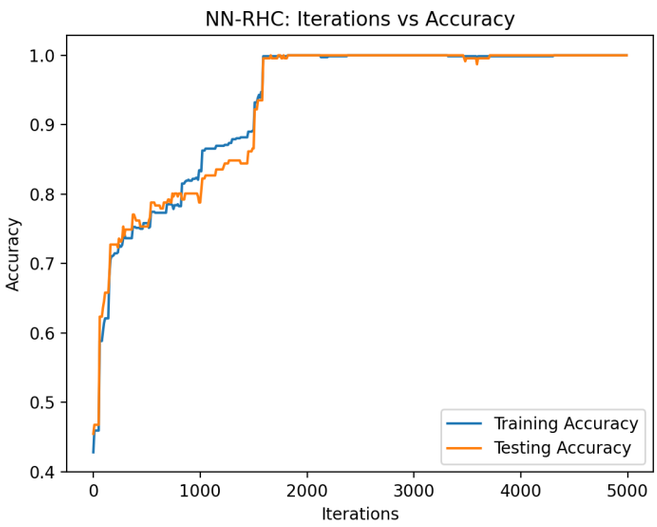
Layers for NN in Assignment 1
The optimal layer combination for this data set was 100 input layer nodes, 4 hidden layer nodes and 1 output layer node for each classification. The activation function is TANH. This is the same I have used for all the algorithms.

Algorithm	Parameter values	Time (in seconds)	Train Accuracy	Testing Accuracy
Back Propagation	-	0.02 s	98.5%	99.99%
RHC	Iterations = 2000	2s	99.8%	98%
SA	Iterations = 2100 Cooling = 0.85	1.7s	99.5%	98.2%
GA	Iterations = 80 Population=100 Mate = 10, Mutate = 30	10s	99.7%	99.1%

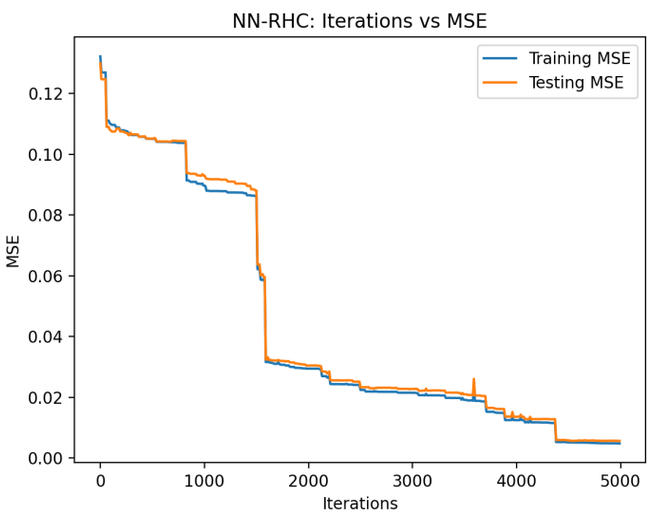
The above results show that though the three algorithms achieved pretty good results, they were comparatively very slow in achieving that accuracy. The running time of SA was a little less, and they converged to almost similar accuracies in 2000 iterations. GA was the slowest, but the number of iterations were least, with highest accuracy.

Lets do a detailed analysis of each algorithm:

RANDOMIZED HILL CLIMBING



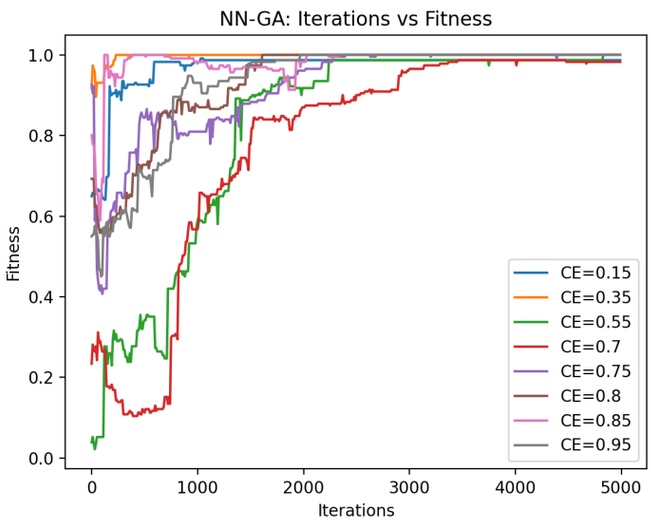
Graph RHC.1



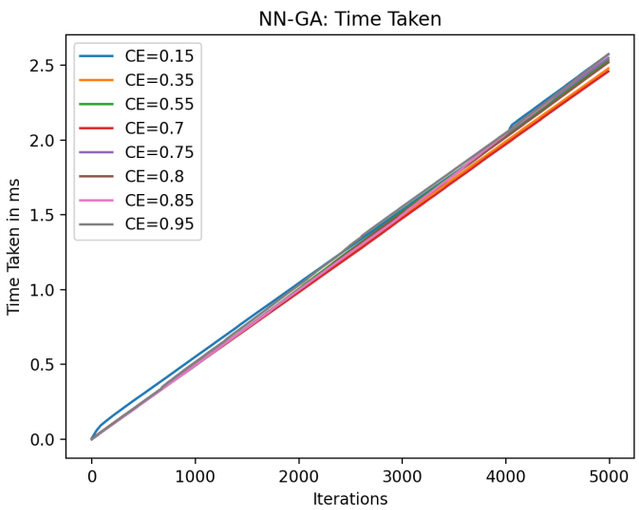
From the figure RHC.1, we can see that the accuracy curve converged quite fast and is almost constant after 1800-2000 iterations. This indicates that the local optima has been obtained, and another peak could not be found.

Compared to the other two optimization algorithms, RHC got almost the same performance as SA. But maybe because of the way the data is distributed, it could not match the performance of back propagation, as it could not find the global optima and is stuck, though performance is pretty good.

SIMULATED ANNEALING



SA.1



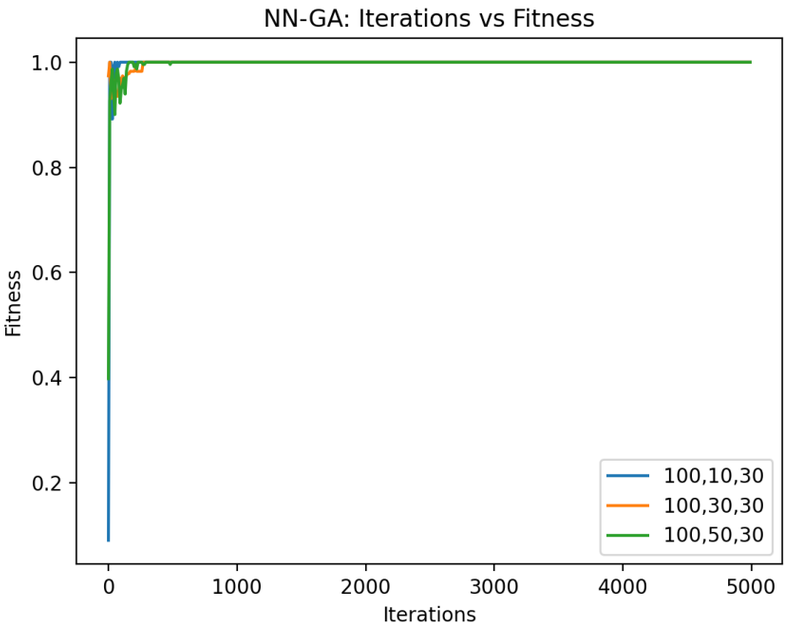
SA.2

The above graphs show the performance of SA with various cooling temperatures. The cooling factor of 0.85 gives the best accuracy, which is quite comparable to the RHC accuracy. The time taken as compared to RHC is a little less, but still not as optimised as the back propagation. Since SA improves the probability of picking up a better neighbour, it performs slightly better. The similarity between the RHC and SA show that both algorithms can find all local maxima, and one of the local maxima is closest to the global maxima.
For Ce < 0.85 and Ce > .85, we can see that the algorithm explored the space more and had less accuracy. The lower cooling rate lead to converge slower and higher one needed more less iterations to converge. Once converged, all the curves became smooth, which indicate that the optima has been found.

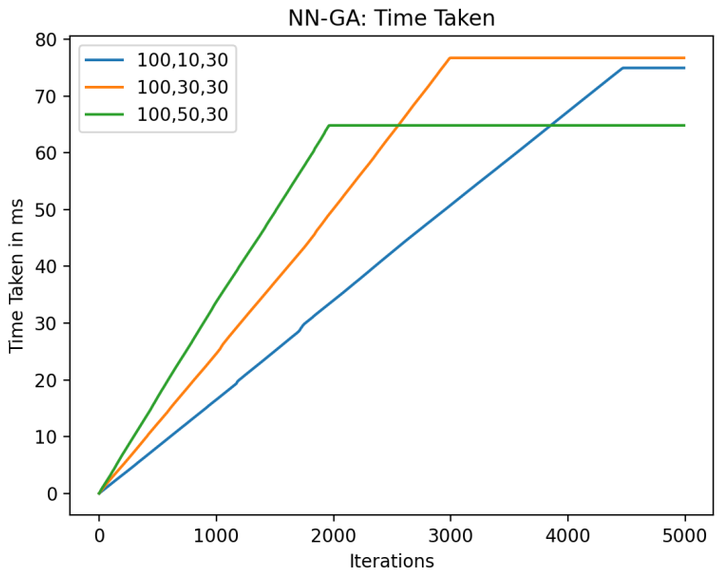
GENETIC ALGORITHMS

For genetic algorithms, I iterated over population, mate and mutate combinations of [100],[10,30,50],[10,30,50]. The graphs below show lines for population 100, mutate 30 and mating values 10,30,50. The best results are obtained for the combination 100,10,30. The blue line below in graph gets covered by the green one, but it converges to the optimal value fastest and smoothens out. The time taken by the combination is slightly greater than the other two, but the accuracy is higher.

As compared to other algorithms, it performed worst in terms of computation time, but best in terms of accuracy. However, the accuracy is just slightly better while the difference in time taken is very high.



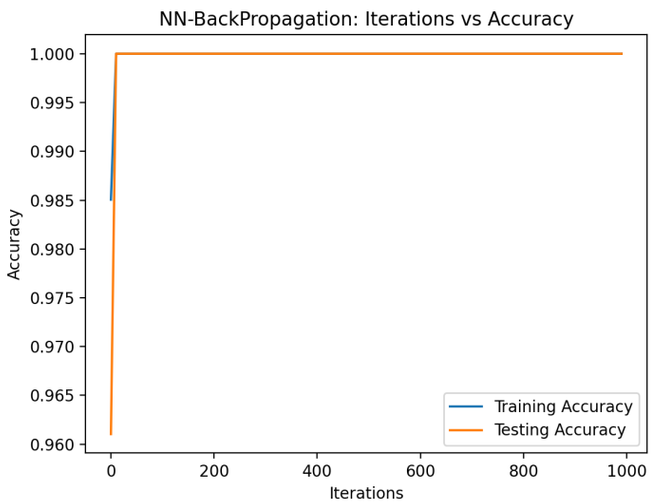
Graph GA.1



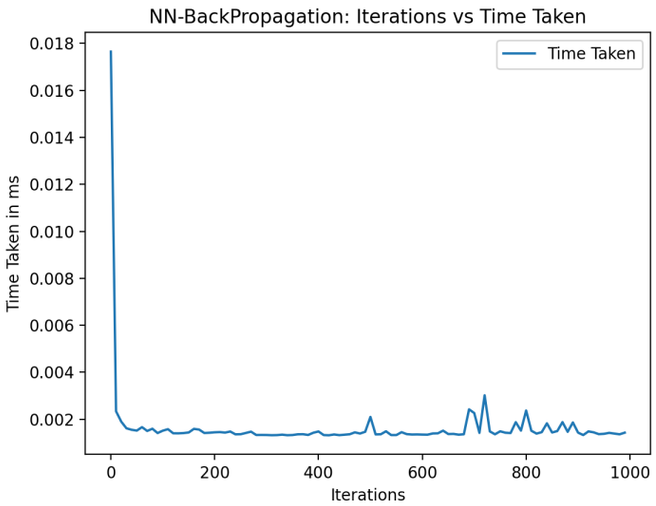
Graph GA.2

The dataset has 18 continuous features, which makes it a huge hypothesis space, hence the algorithm takes a large time to converge.

COMPARISION WITH BACKPROPAGATION



BackPropagation 1



BackPropagation 2

As seen above, back propagation takes the least amount of time to give almost the best accuracy. None of the above algorithms compare to it in terms of time and accuracy both.

Even though, if I had to choose one of the above 3 algorithms, to use instead of back propagation, for this dataset I would choose Simulated annealing.

- Genetic algorithms are not that great because of their high run time. The data set is relatively simple with less amount of data, still it takes about 10 times more time to converge.
- Simulated annealing gives almost perfect accuracy in a the lowest amount of time.

References:

- https://github.com/pushkar/ABAGAIL
- https://github.com/JonathanTay/CS-7641-assignment-2/blob/master/NN1.py - The errorOnDataSet function to calculate MSE.