Name: Het Piyush Sheth **GTID:** 903393644

CS 7641-HW2

Introduction

This assignment deals with Randomized Optimization problems. This report is divided into three sub-sections. The first sub-section gives a brief introduction of the various Randomized Optimizations algorithms that I have used, i.e. Randomized Hill Climbing, Simulated Annealing, Genetic Algorithms and MIMIC. The second sub-section explores the performance of the first three algorithms in determining the weights of a Neural Network. The last sub-section explores the performance of the four algorithms for three different optimization problems. I have executed the above algorithms using the ABAGAIL library with Jython. I have used the Adult Income Data for the Neural Network. It contains 14 attributes with 30,000 instances (approx.) and using these attributes, the aim is to classify whether a person's income is below or above $50,000 per annum.

Randomized Optimization (Section 1)

This broad term encapsulates a set of algorithms that are used to obtain the global optimum of the input function. The advantage of this set of algorithms is that they do not need the function to be differentiable (or continuous) in order to find the global optimum. Below is a brief description of the four optimization algorithms that I analyze in this report.

Randomized Hill Climbing (RHC):

Hill climbing selects a random point on a curve and keeps moving upwards till it finds a maximum point. The issue with this approach is that the algorithm quite often gets stuck in the local maxima and never reaches the global optimum from there. Thus, an optimization of the Hill climbing algorithm is used, known as Randomized Hill climbing. Here, the algorithm works like a normal Hill climbing algorithm but it repeats iteratively for multiple iterations, each time starting with a new random point. This way the probability of it reaching the global maximum increases substantially. For my implementation, I used 5000 iterations.

Simulated Annealing (SA)

Annealing refers to the process of continuous heating and controlled cooling of a material used to strengthen its structure. Simulated Annealing derives its roots from the Annealing process where we first choose a random point and then we evaluate a neighboring point. If the neighbor point is better than the current point then we move to that point but we do not directly discard a point if it is lesser than the current. Instead, we calculate the probability to go to the next point based on an acceptance function. This function is a combination of the initial temperature and the cooling rate. This algorithm would also select a neighboring point that would lead to a slow decrease in probability as it helps in exploring the neighboring spaces. SA reaches the global optimum eventually, but slowly.

Genetic Algorithms (GA)

This algorithm finds its root from the reproduction process where the traits of two parents are combined to make a child (offspring). This child has traits that are a combination of the two parents derived by crossover

and mutation. This process is continued for multiple generations to get the best traits. The idea is that initially, a population is selected and they are mutated to generate a new population. Then, the children with the best features are moved onto the next generation along with their parents. After multiple iterations, the population evolves to the global optimum. Although this method has certain drawbacks that are presented with scaling. When the input increases, the search space increases exponentially.

Mutual Information Maximizing Input Clustering (MIMIC)

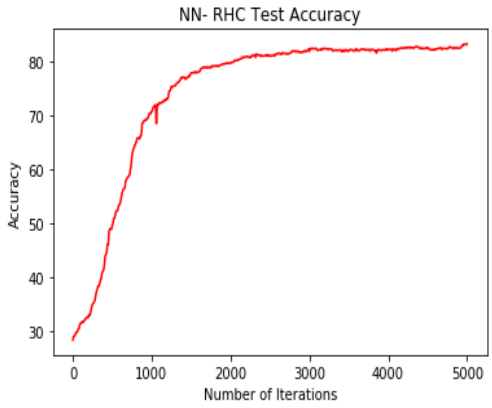
MIMIC uses the structure of the solution space to find the global optimum. It finds this structure using probability density functions. It runs for multiple iterations and each iteration, it finds a better solution space structure, eventually resulting in a global optimum. One of the disadvantages of this method is that the time complexity for it to run is significantly higher than its peers.

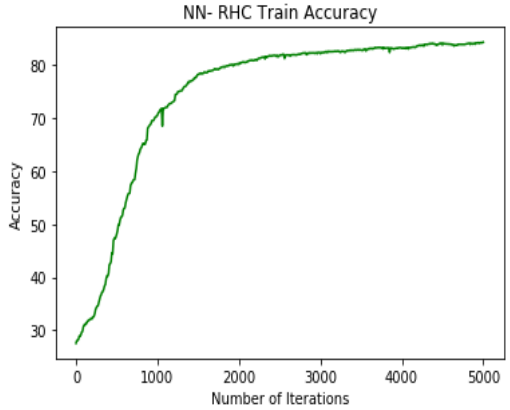
Optimizing weights of a Neural Network (Section 2)

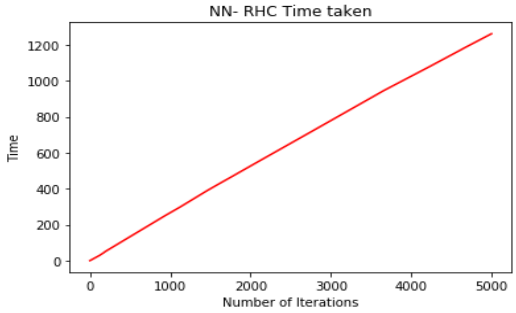
In this section, I am using the Neural Network that I used for the previous assignment and applying the random optimization algorithms in order to get the best weights for the network. I have used three hidden layers as my dataset does not have continuous input and thus it cannot be effectively learned by one hidden layer.

Randomized Hill Climbing

Below are the graphs that show how my model performance improves by optimizing the weights of the neural network using RHC. The best testing accuracy obtained is **83.25%**. For the initial iterations, the accuracy significantly increases and then after a point (approx. **1800** iterations), the accuracy remains near constant and shows only little improvement. This is because the algorithm has probably found the global optimum and the further iterations and leading us back to the same point. This trend is followed for both the training and testing accuracies confirming the correctness of this approach. Also, this is not entirely visible at the zoomed out version of the graphs but if we look further, i.e. the trend in the accuracy at a level of tens of iterations instead of thousands, then we will notice that there is quite a fluctuation in the graph. It is not entirely smooth. This is because RHC is ideally a directed search that starts from random points. The fact that it is a directed search also explains why there are only very few fluctuations. The training time increases with an increase in the number of iterations (as seen in the graph). It took **1200** seconds to run for 5000 iterations.

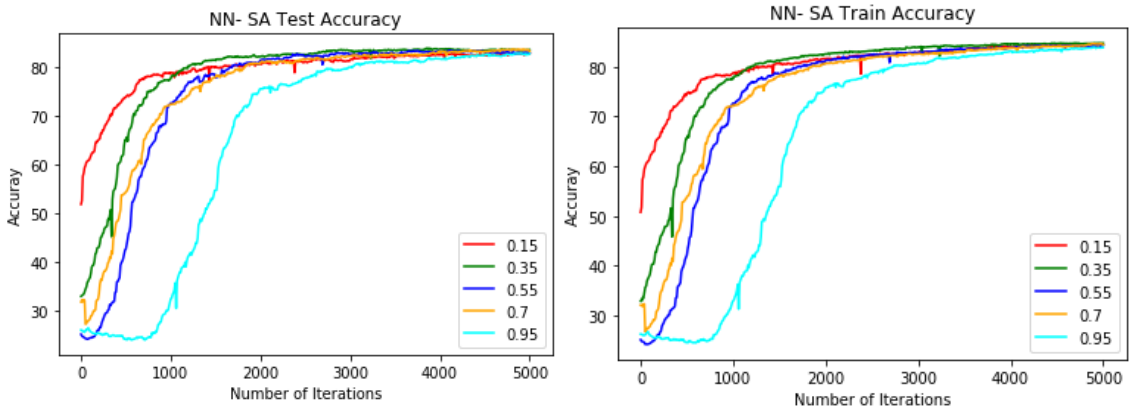


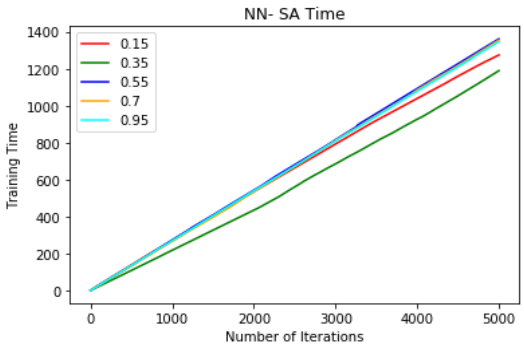




Simulated Annealing (SA)

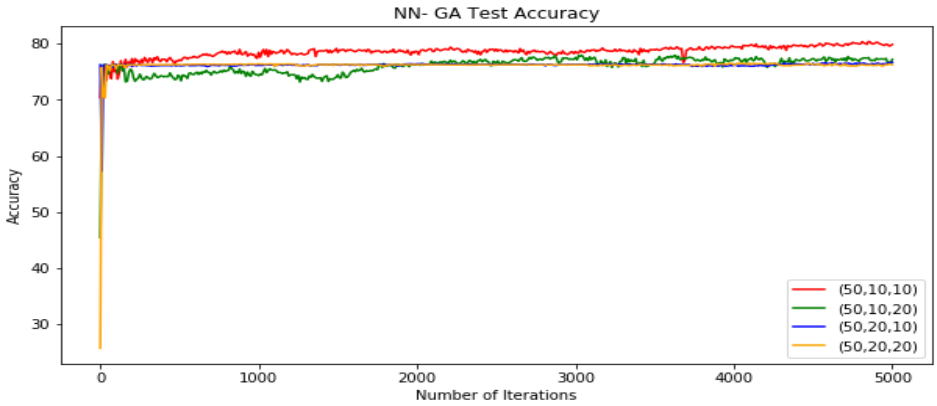
Simulated Annealing takes into consideration two parameters: The initial temperature and the cooling component. I set the initial temperature to **10^10** and for every iteration, I slowly decreased the temperature by taking its product with the cooling exponent. Greater the value of the cooling exponent, slower is the decrease in the temperature. If the cooling component is too high, then the increase in the accuracy is very slow. Training time also depends on the cooling exponent (as seen from the graph). A lower value of the cooling exponent means that the model will take less time to train. The best testing accuracy obtained is **84.93%**. As seen in the graphs, a value of **0.15** gives the overall best training performance for lower iterations. And **0.35** increases steeply as the number of iterations increase, eventually giving a greater accuracy than that of **0.15**. This trend is followed for both the test and train accuracy. Also, one thing to notice is that the highest accuracies are similar to the ones obtained in RHC. This is because for lower values of the cooling exponent (**~0.15**), Simulated Annealing behaves similar to RHC. At the same time, the lowest accuracy is obtained at the highest value of the cooling exponent. This is because for high values of the cooling exponent, the algorithm behaves similar to a random walk. One other observation I can make from the graphs is that upon increasing the number of iterations, after a point the curve becomes flat. This is because upon increasing the number of iterations, the temperature decreases and makes the algorithm more inert.

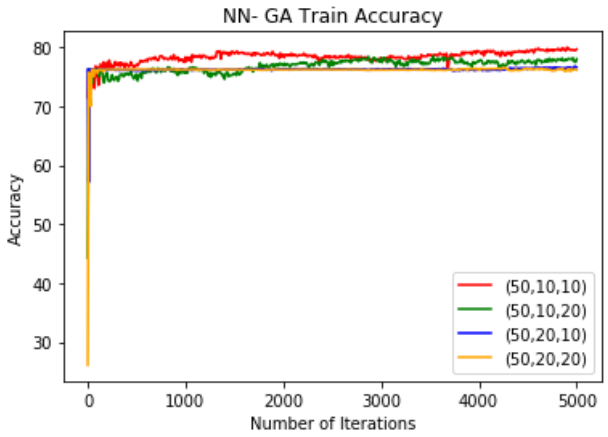


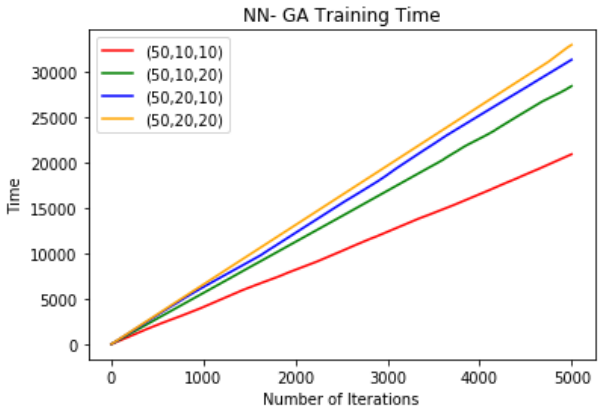


Genetic Algorithms (GA)

The outcome of GA depends on three different parameters: Population Size, Number of instances to be mated and Number of to be mutated. For my analysis, I kept population size as **50** and the number of mated and mutated values in the set **(10, 20)**. After trying out all the permutations of the above parameter values, I got the best testing accuracy of **80.69%** using **(50, 10, 10)**. Also, one interesting thing that can be observed from the graph is that there is a sudden and very steep increase in the accuracy for an initial increase in the number of iterations. This is because we have a huge search space in the beginning and thus we end up evaluating a greater range of NN weights initially. From there, the best of the population moves forward. Now, the flat line right after the initial steep curve is because once we've selected the best offspring and its respective parents from the population, then the further reproductions will result in a similar value of the weights itself as we're mutating the same genes further and further. Also, the training time increases as we increase the number of combinations and mutations, which makes sense as at each step the algorithm would now have to try and test more combinations of the population.







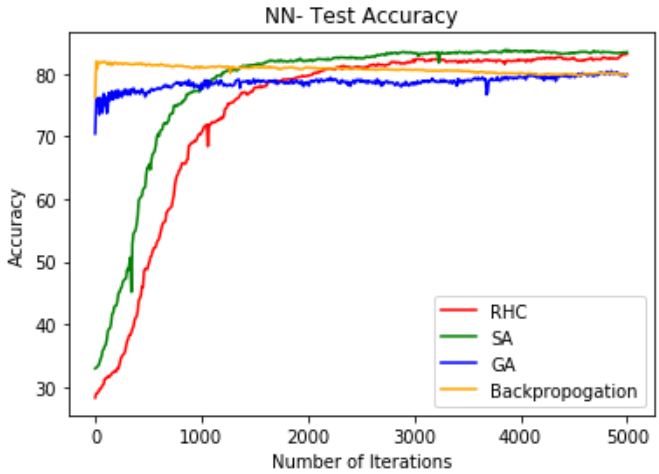
Comparison of the Three Optimization Algorithms

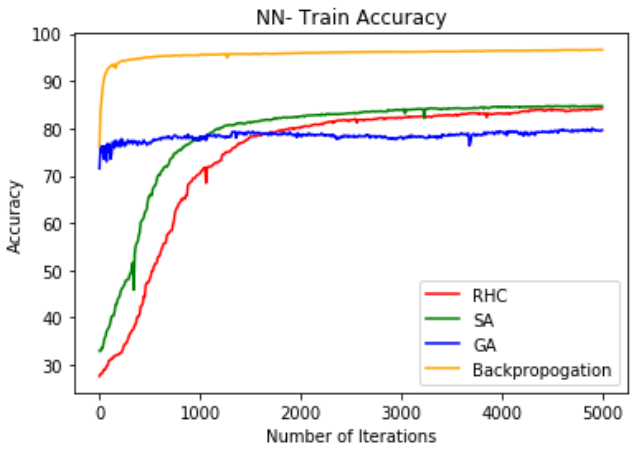
I have tried three different optimization algorithms on the Neural Net and the one with the best performance at **5000** iterations is Simulated Annealing and Randomized Hill Climbing (both have a very similar performance). At the same time both of them have a similar trend in the training time as well. Below are few of the insights that I have derived by comparing all the three optimization algorithms.

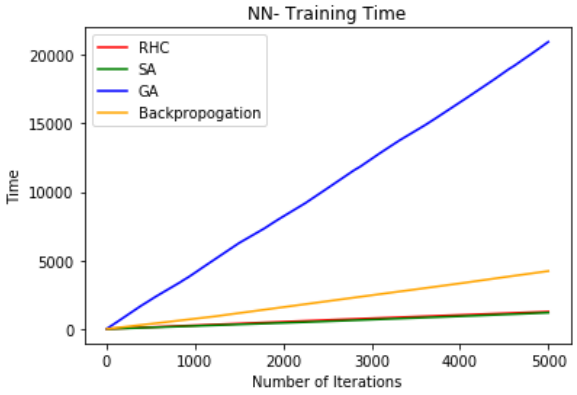
- At the end of **5000** iterations both RHC and SA reach the global optimum. Since SA is a probabilistic algorithm and RHC is a randomized algorithm and they have similar results, we can conclude that the search space doesn't have many local optimums, as if that were the case, then the performance of both the algorithms would have varied a bit.
- Given a choice, RHC would be the best algorithm to choose in this case, as the computation time for RHC is the least as compared to the other algorithms.
- If there was a limit to the number of iterations we can use, then GA would be the best algorithm followed by SA and RHC. This is because GA reaches a significantly higher accuracy for very low number of iterations, whereas SA and RHC do not reach that until **1000** and **1400** iterations respectively.
- For lower iterations SA performs better than RHC because SA uses a probabilistic model as opposed to the random model used by RHC. Although eventually, for 500 iterations both the models perform similarly.
- The training time is significantly higher for GA as we increase the number of iterations. This is because in GA, we initially start with the whole population and find the best one from there as opposed to starting with a random point as done by the other two algorithms.
- Thus, the final accuracies obtained are as follows: **84.93% (SA), 83.25% (RHC), 80.69% (GA)**

<u>Performance of the three optimization algorithms with respect to Backpropogation.:</u>

- We can see that backpropogation gives a way better train accuracy as compared to the other three algorithms but doesn't show that great a performance in the for the test accuracy. This is because using backpropogation for optimizing the weights of the model is actually resulting in overfitting of the model.
- Overfitting occurs because we have a very large training set and this is only a binary classification problem. Thus, after a certain number of iterations, the backpropogation model just starts learning on noise rather than patterns. Thus, if we notice, the accuracy is actually better initially for lower number of iterations for backpropogation.





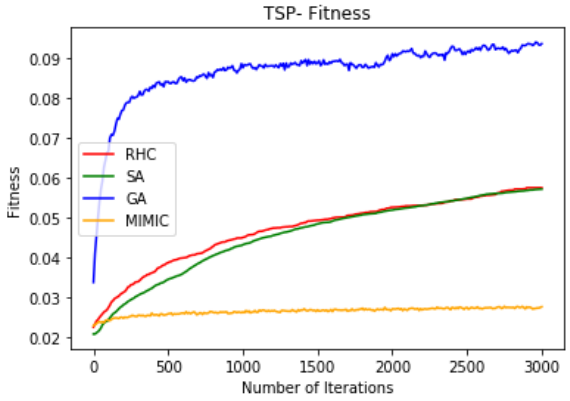


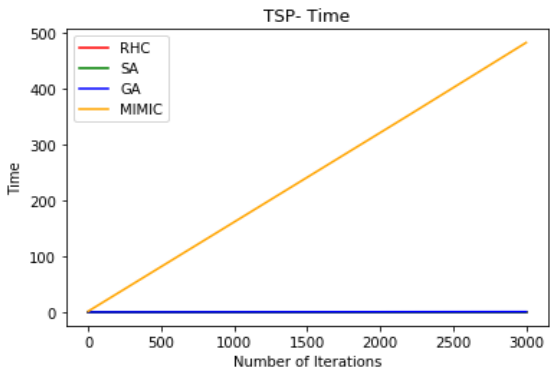
Optimization Problems (Section 3)

Next, I implemented the four optimization algorithms described above on three different optimization problems, namely, Travelling Salesman Problem, Flipflop and Continuous Peak. I chose these three problems because they gave me different results for the different optimization algorithms. For example, Flipflop optimization problem does not have a constant specific pattern and thus GA does not work very well for it. On the other hand, GA works very well for the Travelling Salesman Problem.

Travelling Salesman Problem

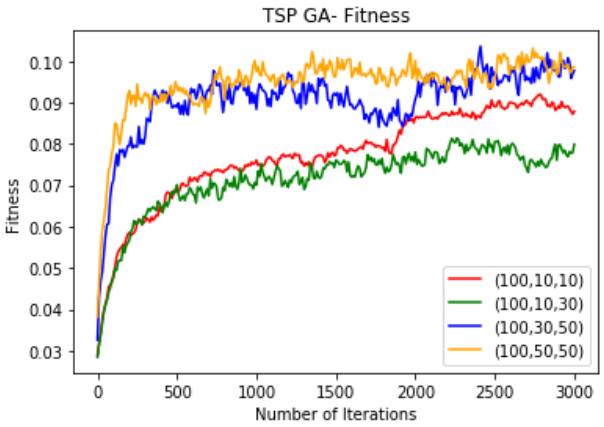
In this problem, we are given a list of cities and the distance between each pair of cities as the input. We have to find the shortest possible route starting from one city, visiting every city exactly once and coming back to the city we started from. This problem is also one of the most famous NP-hard optimization problems. I used the four algorithms mentioned above to optimize this problem. Except RHC, I used several hyper parameter combinations for the other three algorithms and took the average of all the different results. Here, I took the average of all the permutations and not the best combination because I wanted to compare the performance of the algorithm as a whole and not one specific case of the algorithm. Upon plotting the performance of the various algorithms, we can see that GA outperforms the other three by a great margin. This is because GA can detect a constant specific pattern very easily. Also, GA doesn't require large number of iterations because it makes use of mutations and crossovers. On the other hand, we can see that RHC and SA are constantly increasing with an increase in the number of iterations and have not flattened out like MIMIC. This means that there are several local optimums in this problem space and thus both RHC and SA are getting stuck in them. I believe that given more iterations, both the algorithms will eventually reach to the global optimum. As for the time taken, MIMIC takes exponentially larger amount of time as compared to the other three algorithms.

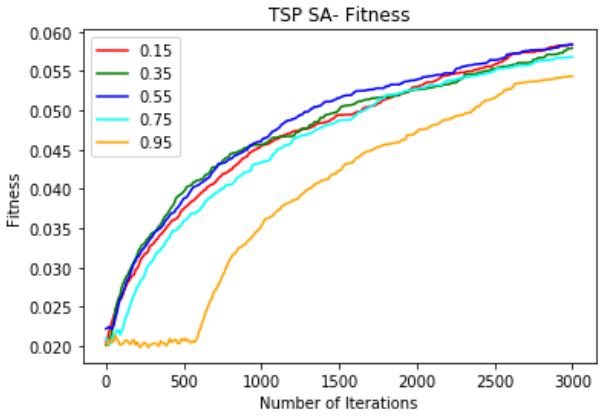




I also decided to plot the fitness variation of GA and SA with respect to the various parameter values. The two things I observed are:

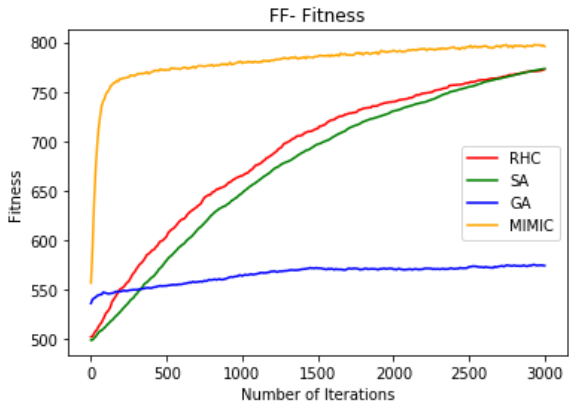
- For GA, **(100, 50, 50)** seems to show the best fitness. This makes sense because there are many underlying patterns to decode in the search space and thus we need a greater number of combinations and mutations to reach the global optimum.
- For SA, the fitness values do not vary much for the lower values of the cooling rate as opposed to the cooling rate of 0.95. This is because the search space is quite huge with several local maxima (reason explained earlier) and the value of **0.95** is equivalent to a random walk, thus giving poor results.
- One interesting thing that I found is if, to optimize a problem, we see that the greater number of mutations and combinations work better using GA, then the search space has a definite pattern and thus many local maxima's. Consequently, SA and RHC do not perform that great for lower number of iterations.

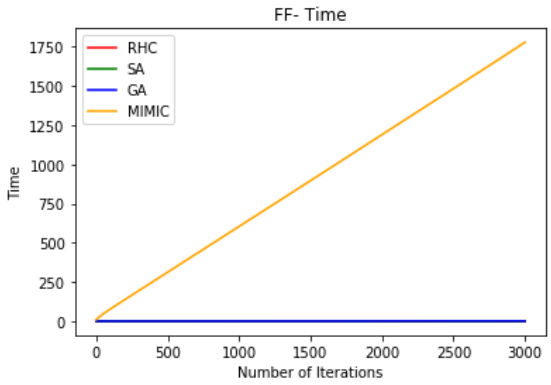




Flip-Flop Optimization

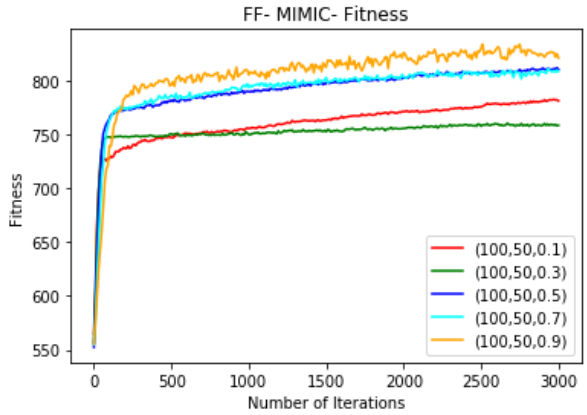
In this problem we are given a fixed length binary string as an input and we have to find the string with the maximum number of alternating bits (termed as flipflop). Here, I used the four algorithms and tried a variety of parameter values which I show in the graphs. By looking at the graphs, we can see that MIMIC has the highest fitness score and it achieves this at a very lesser number of iterations. At the same time, RHC and SA have their fitness score increasing as we increase the number of iterations and they haven't yet reached the global optimum indicating that we can get a better fitness score for those algorithms by increasing the number of iterations. Also, one very interesting thing from the graphs is the performance of GA. While GA outperforms the others in the previous optimization problem, it performs the worst in this one. It is because the algorithm has a tendency converge to the local optimum rather than the global optimum for this problem. That is, it prefers short term fitness over long term fitness. Mutating one value in the binary string would make the fitness score less and thus GA cannot perform well. Although, MIMIC has the best fitness score, the time taken to execute is exponentially greater than the other algorithms, thus one could prefer RHC or SA over MIMIC if the emphasis was on computational time (as those two algorithms would give similar fitness score with greater number of iterations and significantly lesser time).

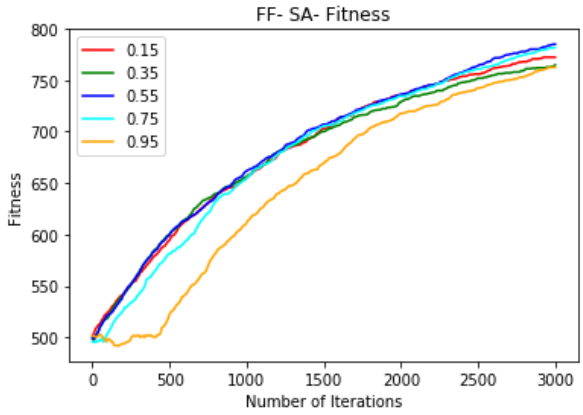




I also plotted the fitness for the various parameters I tried for MIMIC and SA as they outperformed the other two algorithms. We can see that:

- MIMIC performs the best for the parameter with the highest value. This means that the search space is vast and greater number of operations is required for MIMIC to give better results.
- On the other hand, SA performs equally for the low valued cooling rates and the performance is significantly less for the highest cooling rate **(0.95)**. This can also be proven with the insight derived above that the search space is vast and it has several local optimums which is why cooling rate of **0.95** (performs similar to a random walk) does not give good results.

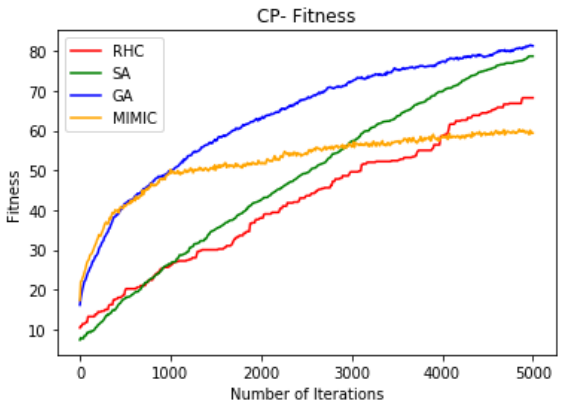


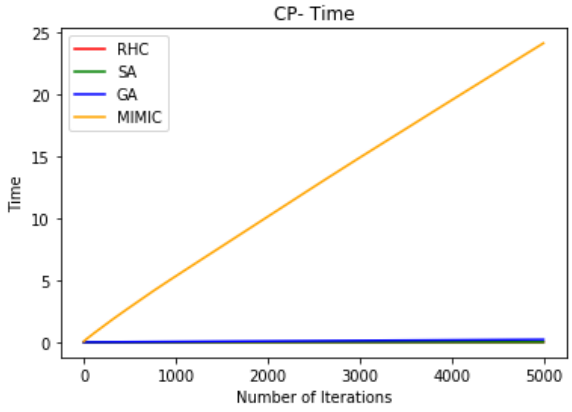


Continuous Peaks

In this problem, the search space has many local optima and our task is to find the global optima in the search space. Just by looking at the problem, we can say that a randomize approach would not perform as well as the probabilistic approach and this is reflected in the graph as SA is performing better than RHC. Although, initially RHC performed better than for the first **100** iterations because SA uses a probabilistic approach which makes it better over time. We can see that GA performs the best amongst all algorithms and it increments steeply initially and the slope reduces as the number of iterations increase and this is because it starts with the whole population and finds the best within it (just like bottom up).

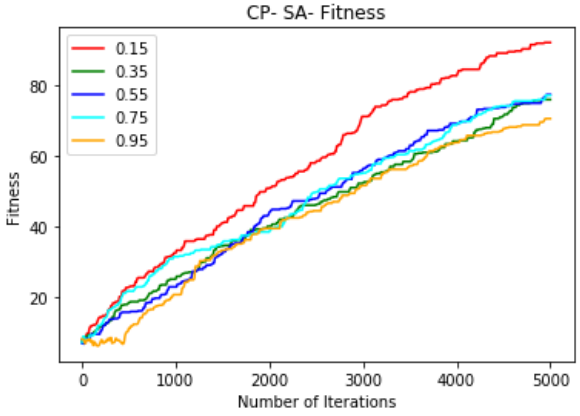
For the first **1000** iterations, MIMIC and GA perform similarly but MIMIC takes a significantly larger computational time thus making GA better.

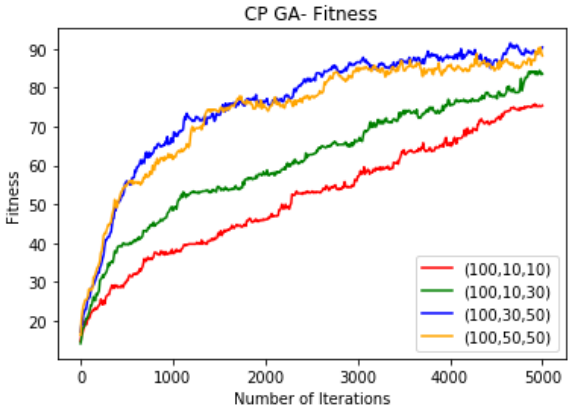




I plotted the variations in the fitness scores for different parameters for SA and GA as they have outperformed the others. We can see that:

- The lowest value of the cooling exponent gives the best accuracy in SA. This is because a lower exponent penalizes the temperature lesser and thus does not behave like a random walk (which higher values do). This makes sense as the problem presented has many local maxima's.
- For GA, the worst performance is with the lowest mutation and cross over values. This is because our search space is huge with many local optimums thus having lower values converges the function to a local optima. Consequently, the higher values of mutation and cross over perform better.





Conclusion

I have tried the four optimization algorithms for three different optimization problems. Upon analyzing the results, I realized that no one algorithm is an absolute superior to the other and the results depend on the type of the problem we are presented with. The table below indicates the best performing algorithm for each optimization problem and the reason the algorithm is the best one.

Optimization Problem	Best performing algorithm	Explanation
Travelling Salesman Problem	Genetic Algorithm	GA can detect constant patterns very easily. This problem has several patterns in its search space.
Flip-flop Optimization	MIMIC	MIMIC uses the structure of the solution space using probability density functions, which is important in this

		problem as we need to find alternate bits in a string, which requires the algorithm to explore the search space probabilistically rather than randomly
Continuous Peaks	Genetic Algorithm/Simulated Annealing	Even though GA has a slightly greater fitness score, it takes significantly longer than SA for computation, thus making SA better overall. This is because the solution space has several local optimums and thus in order to not get stuck in one of them, we need to approach the problem probabilistically.

Also, it is essential to summarize the algorithms and when they should be used (as observed from the analysis).

Algorithm	Number of input parameters	Traits
Randomized Hill Climbing	**0**	Works well for small cost problems that have a simple structure. Also performs well for time sensitive applications
Simulated Annealing	**2** (Initial Temperature, Cooling exponent)	Works well for small cost problems that have a simple structure. Also performs well for time sensitive applications
Genetic Algorithms	**3** (Initial Population, Crossover, Mutation)	Works well for problems that require a complex structure and the emphasis is on the fitness rather than the computational time.
MIMIC	**2** (Samples, Keep)	Works well for problems that require a complex structure and the emphasis is on the fitness rather than the computational time.

This concludes the analysis for the use of various optimization algorithms for optimization problems. (The conclusion for the use of these algorithms to find the best weights for a Neural Network can be found above, right before the Optimization Problems section).

References

1) Jonathan Tay - Github

2) UCI ML Repository- Adult Dataset

3) ABAGAIL

4) Jython

5) Forest-Mitchell - What Makes a Problem Hard for a Genetic Algorithm?