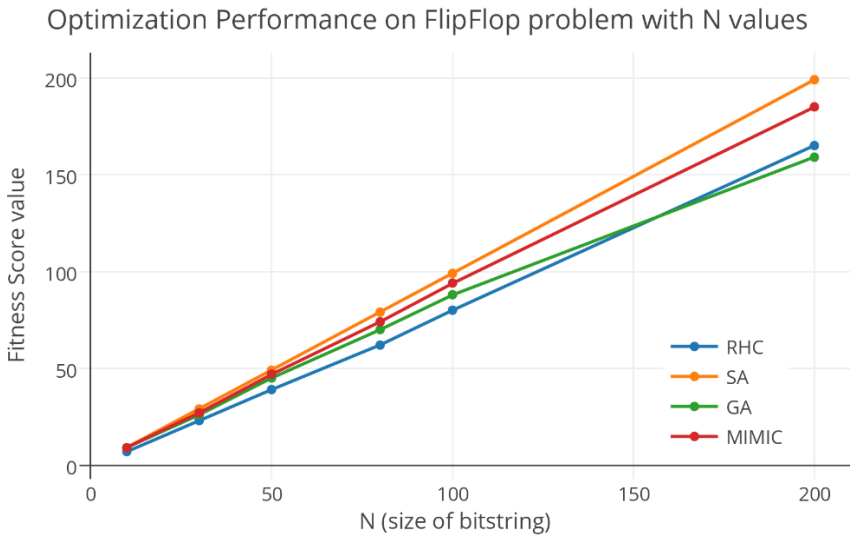
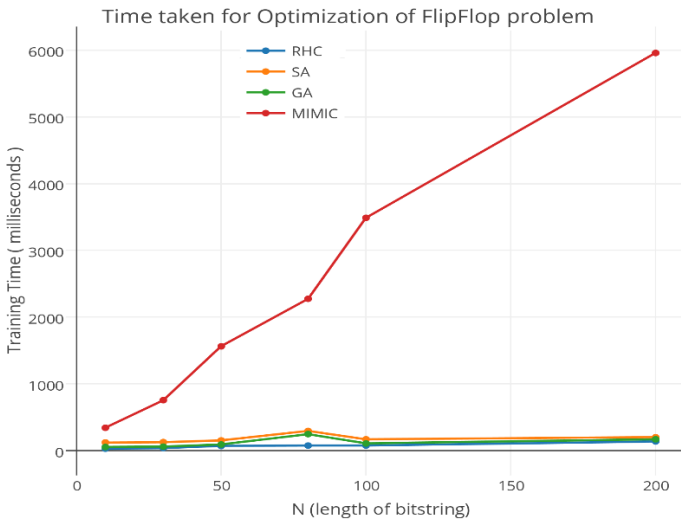
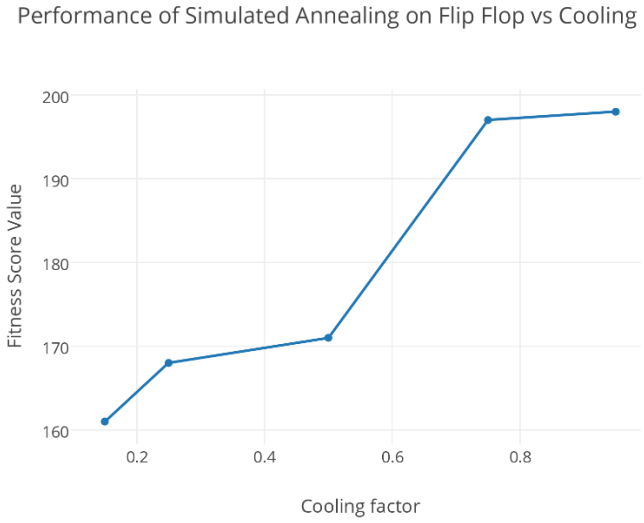
Jassimran Kaur Somal ***Assignment2***
GT-ID : jkaur9

Simulated Annealing

Flip Flop function returns the number of times bits alternate in a bitstring, For example: “0011” would return 2. “0101” would return 4. The optimal solution produces a constantly alternating bitstring with a return value equal to the length of the string. Initially I ran Flip Flop problem on varying length bitstrings to find a suitable bitstring length which highlights the differences among all algorithms and I selected N = 200







The ideal cooling rate cannot be determined beforehand, and should be empirically adjusted for each problem domain and for this experiment, after running experiment with different values of cooling factor, I used 0.95. For

the small value of N, varying starting temperature had little influence on the results and hence I stuck to 100. Below table highlights the parameter values used for analyzing the Flip Flop optimization problem.

Algorithm	Iterations	Parameters, N = 200	Average Result of 10 trials	Total Time (milliseconds)
Randomized Hill Climbing	200000		163	140
Simulated Annealing	200000	Initial Temperature T=100, Cooling factor C=0.95	198	203
Genetic Algorithm	200	Population Size=200, MA=100, MU=20	159	172
MIMIC	200	Samples=200	185	6297

Simulated annealing got the optimal solution. Since it is known to work best in a small discrete space where neighbors and local optima are close by, and the flip flop function has a large number of local maximums. Hence simulated annealing can easily get out of these local maximas, whereas Randomized Hill Climbing might converge and get stuck at local maximas as it will often get to a bitstring where changing any of the bits would result in no gain. MIMIC takes maximum time among all the algorithms to run so far and with large number of local maximums of the same height, MIMIC results in a near uniform distribution, which does not improve much upon previous iterations. But simulated annealing is very fast for its level of performance compared to MIMIC.

Genetic Algorithm

TravelingSalesmanProblem: In traveling salesman problem (TSP), a salesman needs to find the shortest path to visit all of the N cities (nodes in a graph with N vertexes. This problem is difficult to solve due to the fact that there are N-1 factorial possible routes that the salesman may take to complete the trip. TSP is an NP complete problem that aims to find a cycle of shortest length on a graph. It's NP hard and not easily crack-able in polynomial time, hence it's an excellent candidate for randomized optimization. Since the fitness function is the minimization of distance, hence to adapt it to maximization strategy, fitness function is derived from RouteEvalautionFunction which maximizes the inverse of distance as shown below:

```
fitness = maximize( 1 / distance )
```

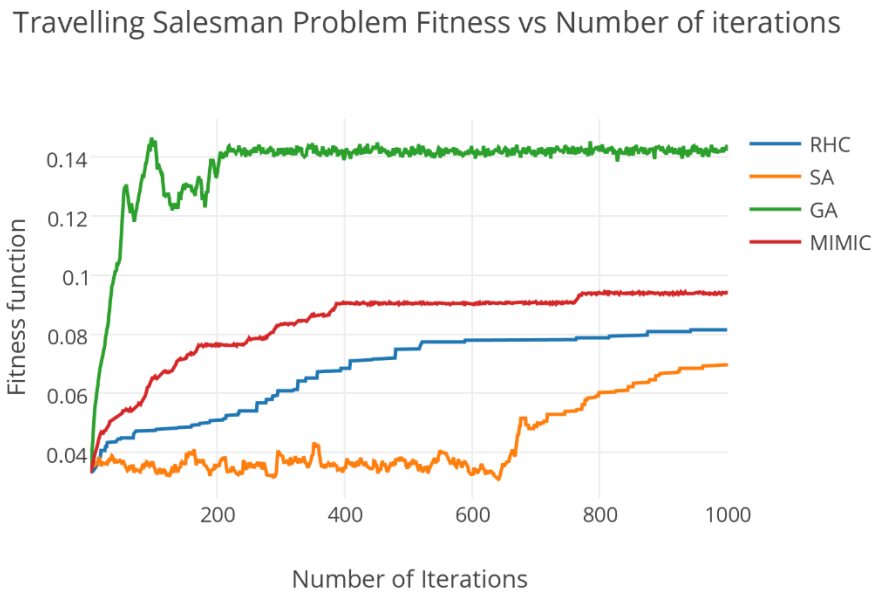
I ran the following algorithms on a fixed set of 50 points:

Algorithm	Iterations	Parameters	Average Result(10 runs)	Total Time (milliseconds)
Randomized Hill Climbing	20000		0.1165	25
Simulated Annealing	20000	T=1E12, C=0.95	0.1228	28
Genetic Algorithm	1000	S=200, MA=150, MU=20	0.1653	424
MIMIC	1000	S=200, K=100	0.0916	16159

The same algorithms were ran till convergence using the same fixed 50 points as shown below:

Algorithm	Max Iterations	Parameters	Average Result(10 runs)	Total Time (milliseconds)
Randomized Hill Climbing	200000		0.0358	<1
Simulated Annealing	200000	T=1E12, C=0.95	0.0387	<1
Genetic Algorithm	200000	S=200, MA=150, MU=20	0.1637	61001
MIMIC	200000	S=200, K=100	0.1085	66418

It is evident that genetic algorithms outperforms all of the other randomized optimization algorithms. It can be attributed to the fact that the genetic algorithm works with entire sets of potential solutions, and only continues with ones that are slightly altered versions of those that did the best out of the previous generation. This in turn makes every further cross over very effective leading to good GA performance. It is noteworthy that genetic algorithms took marginally more time than randomized hill climbing and simulated annealing to complete, however MIMIC as usual was the most time-consuming algorithm.

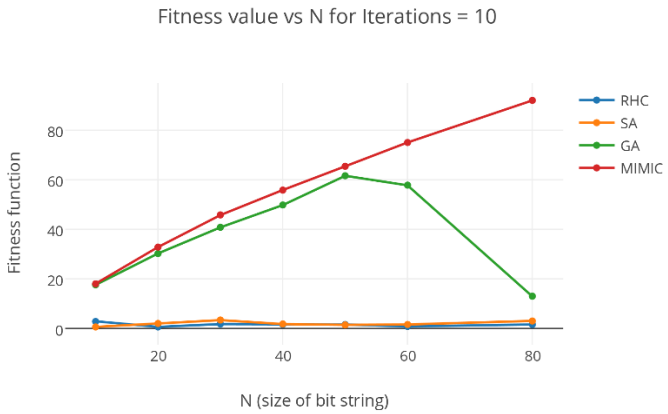


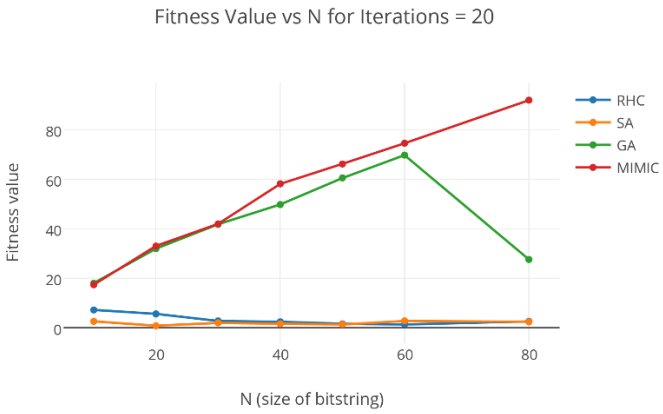
RHC, SA did poorly because after starting with one possible solution and so many ways to branch next, the entire search space becomes too large for them to explore. MIMIC on the other hand looks at the probability that the true maximum is in an area. The search space for this function is very large, and inputs are all independent of one another and there is no underlying structure to the problem that can be exploited to converge to optimal outputs more quickly, thus it might take a lot of time for MIMIC to converge to optimal solution.

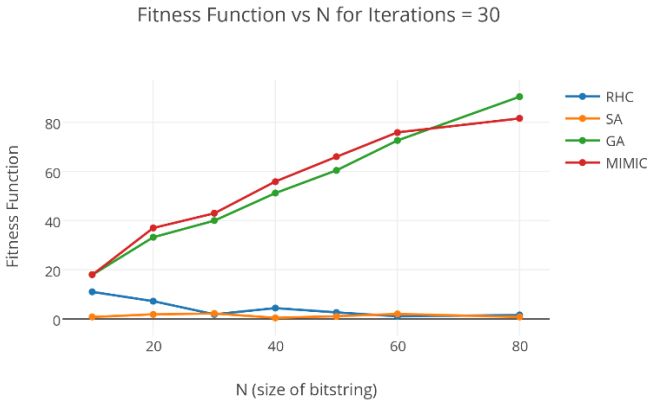
MIMIC

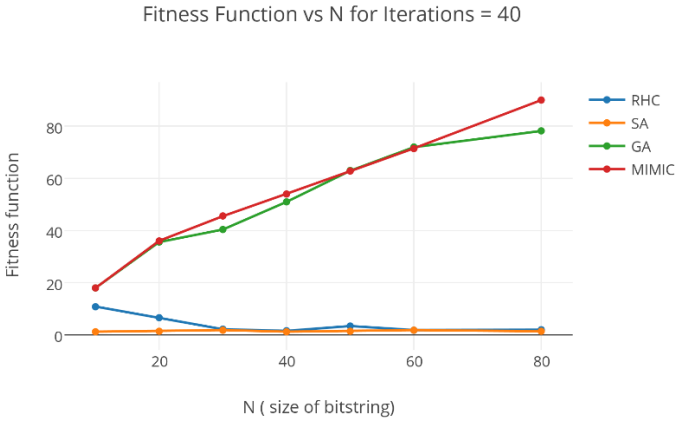
Four Peaks Problem: The four peaks problem is a classic optimization problem and has a total of four maxima's (4 peaks) out of which two are global maxima and two are suboptimal local maxima. Four peaks function takes two inputs: (1) a bitstring of length N which represents the size of the input vector; (2) the size of the basin of attraction T of local maxima's.

I experimented with different values of N but T was always equal to 10% of N because for large values of T, this problem becomes increasingly more difficult because the basin of attraction for the inferior local maxima become larger.









I also ran the algorithms with the following parameters on a FourPeaks problem of N=80, T=N/10 to check performance of each algorithm in attaining Optimal results.

Algorithm	Iterations	Parameters	% Suboptimal Results (Value N = 80)	% Optimal Results (Value 2N-T-1 = 151)	% Other Results
Randomized Hill Climbing	20000		100%	0%	0%
Simulated Annealing	20000	T=1E11, C=0.95	65%	35%	0%
Genetic Algorithm	1000	S=200, MA=100, MU=20	0%	0%	100%
MIMIC	1000	S=200, K=5	20%	70%	10%

MIMIC did very well with almost 70% success rate in attaining the optimal maxima. With the exception of genetic algorithm, all of the algorithms used generated either the suboptimal or optimal solutions but with more iterations even GA would have reached some maxima. As there are only four local maxima, each spread a fair distance apart, far beyond the scope of the neighbor function given to RHC, hence once stuck at suboptimal local maxima, RHC never gets out and always produces suboptimal results. SA wasn't very successful either but every once in a while it was able to break out of local maxima's and find global maxima by injecting just the right amount of randomness into algorithm to escape local maxima early in the process. However Genetic algorithm

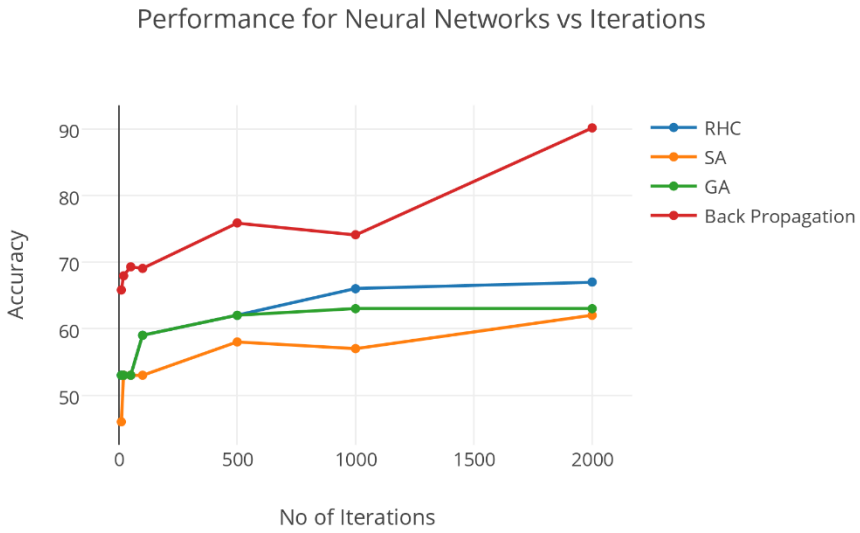
failed to find any of the local maxima with 1000 iterations but given a lot of additional iterations, it did converge but took comparatively more computation time as compared to other algorithms.

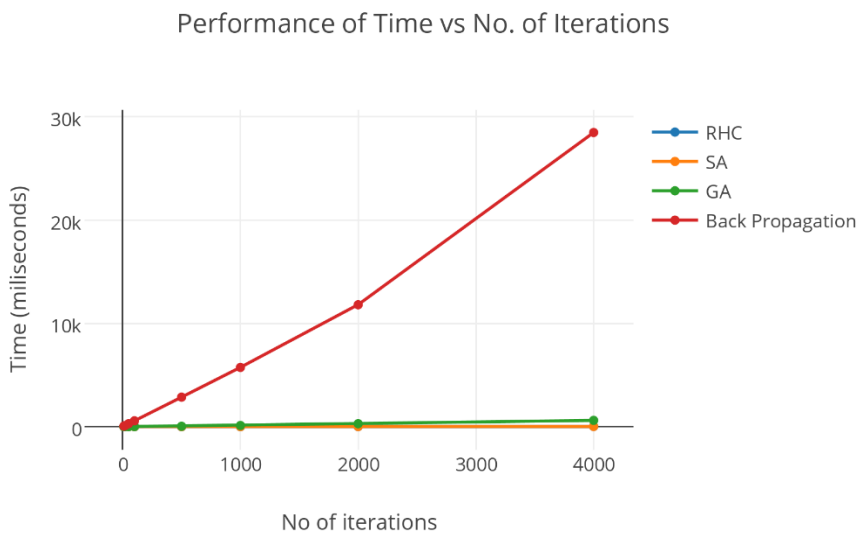
MIMIC performed far better than any of the other algorithms and is effective because it requires the least number of iterations to produce a good result among all the four algorithms. It creates a distribution of the most likely locations for maxima's. By repeating this process and sampling from the previous iteration's distribution, it is increasingly likely to find better maxima's. Suboptimal local maximums are removed as better ones are found, further narrowing the search, till it finds absolute maxima. However, it must be noted that MIMIC's runtime is significantly longer than RHC, SA, and even GAs, due to the added complexity per iteration.

Neural networks:

I used the Wine dataset I used in the Supervised Learning assignment for using various randomized optimization for finding weights for neural networks instead of using backpropagation. All of the results shown in this analysis are compared to the results from the neural network experiments from my Supervised Learning assignment as well. Wine data set has 12 inputs and 1 output and hence I created network with 12 input layer and 1 output layer respectively (I used wine quality better than 5 as good and less than or equal to 5 as bad to have binary output). 5 hidden layers were used for the experiments because it maximized the overall accuracy without making the structure too complex. With any value less than 5, network wasn't good enough to describe the target concept, but with too many, the network became too complex and thus didn't converge fast enough, both of which reduced the final classifier's accuracy. 5 provided that right balance for accuracy and performance time-wise.

I took average of few different trails to record the results of these algorithms.





The training accuracy most algorithms seem to increase with the increase in number of iterations performed, backpropagation did well even with very small number of iterations but other randomized optimization algorithms didn't made a very good start. They started catching up with increased iterations. Thus in order for a randomized optimization algorithm to approach the percentage accuracy achieved with back propagation, a significantly higher number of iterations will be required. May be this is not the best problem to be solved by randomized guessing which is mostly done by these algorithms however back propagation is using calculus to calculate better weights. Also, it must be noted that I didn't normalize the data because I didn't normalize it in my first assignment either. So in order to have a well suited comparison, I used exact dataset I used for backpropagation algorithm. Normalizing the data mighty result in better performance. RHC and SA didn't perform well and I hypothesize that multiple well-spaced local maxima's might have made these algorithms not so useful. Genetic algorithm performed in a very non-coherent way.