Popularity of Online News Article

There are a lot of articles published every day as news, articles or blogs. Some of these articles get very popular and gather a lot of audience as it is shared through social media. There are other articles which are not as popular and some that are good enough that some people might read it and share but, do not create as much buzz in the social sphere. This problem has been devised to classify the popularity of online news articles into 'Unpopular', 'Neutral' or 'Popular' based on the number of shares on social media.

The ability to predict which article become popular and which ones never make it to the top seem challenging and appealing at the same time depending on whether the person is the author, reader, distributor or ad hosts. Authors would like to know if their article is going to be read by a lot of people or just barely noticed by others. Ad distributors could make a lot of money by hosting their ads on popular articles as they are read by millions of people which could help their bottom line. Sometimes, it so happens that smaller sites get a lot of traffic because of a single article that is read by millions of people can cause the site host to implode.

Hence, being able to predict the popularity of news articles beforehand seems challenging but, has myriad of applications. A lot of variables effect which articles are popular and which are not. It could be based on certain geographical region, certain cultural, economic, political, factor but, most importantly the content and style of the article itself. However, in the Mashable dataset that has been downloaded the focus is on the content of the article itself rather than other extraneous factors.

This dataset is one of the datasets collected from UCI Machine learning repository. It was collected and donated by K. Fernandes et al who have used it for their own research to predict the popularity of online news article based on the number of shares in the social media like Facebook and Twitter. The dataset originally was used for regression modelling but, has been broken down into three bins and converted to a classification problem for this assignment.

The dataset was collected over two years and has been scrubbed to be used for various machine learning algorithms. The dataset contains a set of 61 attributes out of which 58 of the attributes are predictive, 2 are non-predictive and 1 is a field goal which is the number of shares in the social media. Even though, researchers used this dataset to do a two way prediction between popular and unpopular dataset I have found that the data is not linearly separable. Instead, there are a lot of dataset that are neither popular nor unpopular hence, I have devised a third class which are articles that are neutral in terms of numbers of shares on the social media.
The following are some of the attributes of the dataset:

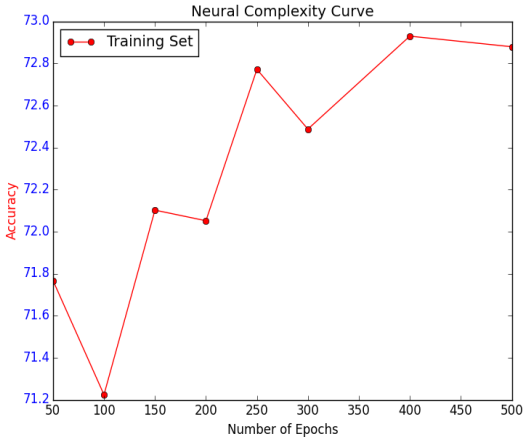
url, timedelta, n_tokens_title, n_tokens_content, n_unique_tokens, n_non_stop_words, n_non_stop_unique_tokens, num_hrefs, num_self_hrefs, num_imgs, num_videos, average_token_length, num_keywords, data_channel_is_lifestyle

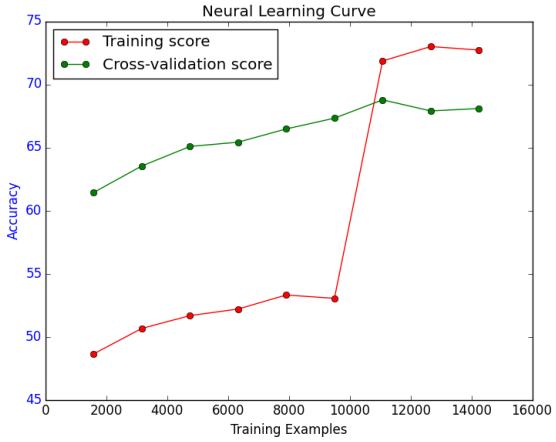
Each one these is taken from various articles published on the Mashable website. I have included two examples below:

> *http://mashable.com/2013/01/07/astronaut-notre-dame-bcs/, 731, 9, 531, 0.503787878, 0.999999997, 0.665634673, 9, 0, 1, 0, 4.404896422*
>
> http://mashable.com/2013/01/07/beewi-smart-toys/, 731, 10, 370, 0.559888578, 0.999999995, 0.698198195, 2, 2, 0, 0, 4.359459459

The target value is the number of shares in the social media which ranges from 1 to 800K so, instead of using the raw values I used standard normalization to fit all of the shares into a smaller range which goes from -0.29194 to 72.23. Hence, a demarcation has been created where articles that have standard shares below -0.225 are unpopular and above 0.1 are popular and those in between are neutral.

Previously, for assignment 1, MultilevelPerceptron from Weka was used to run the neural network with number of hidden nodes as 30. The number of hidden layer was chosen based on the number of input attributes which is 57. This classifier was run for as a cross-validation test and the max score was around 68%. We can also see that the curve was over-fit when the number of training examples was increased from 10,000 to 12000.

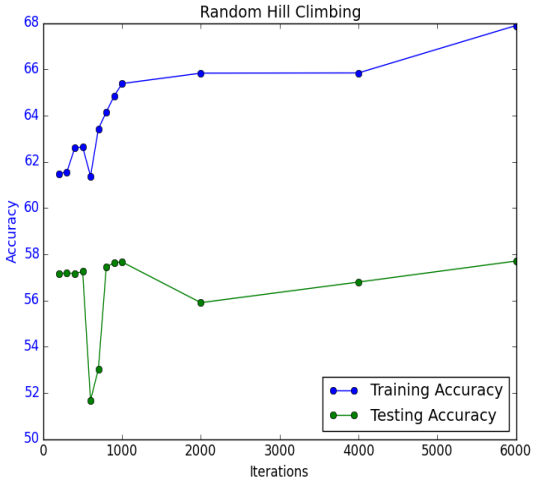


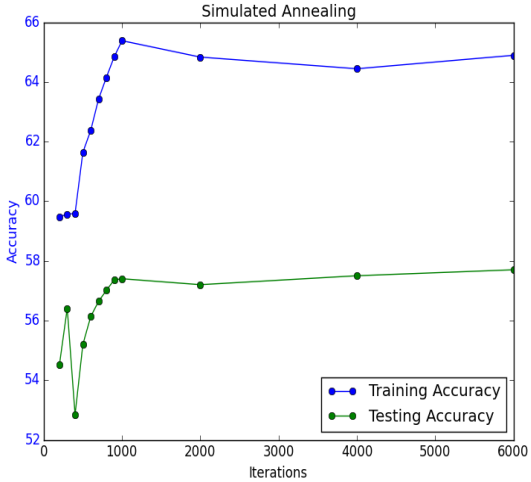


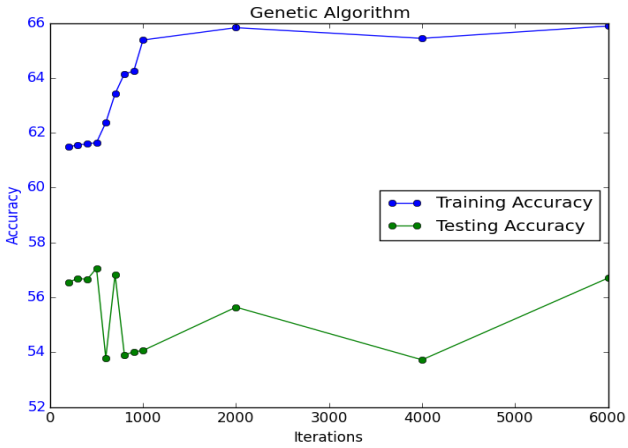
Learning Curve and Complexity Curve from Assignment 1

In order to optimize the weights on the Neural Network three different random optimization techniques were used instead of the backpropagation used in MultilevelPerceptron of Weka. ABAGAIL project was used to run the random optimizations on the same network with 58 input nodes, 30 hidden nodes and 2 output nodes.

Initially, the number of iterations were changed on the various optimization algorithms while keeping the data set the same. The iterations were changed from 200 all the way to 6000 in order to see if the accuracy score of the neural network would improve. We can see the accuracy score of each of the optimizations below:

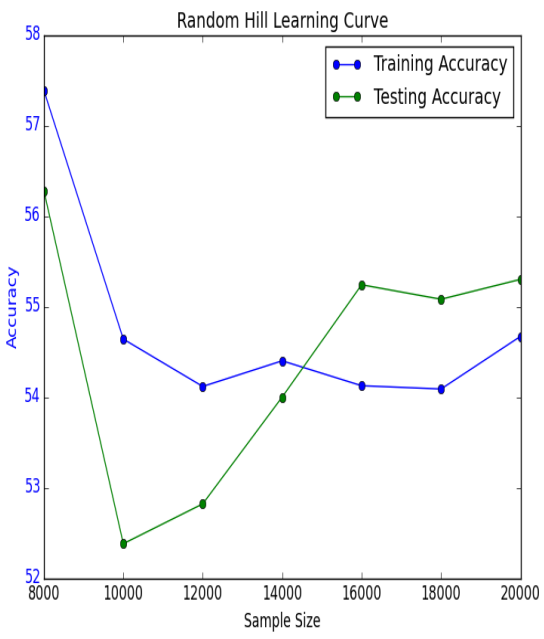


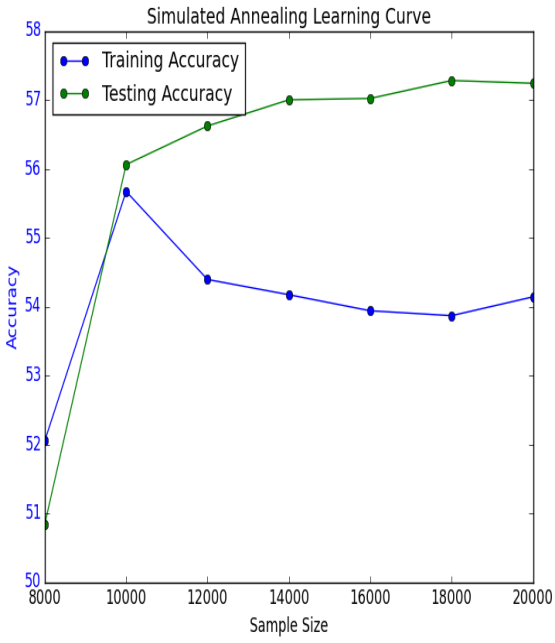


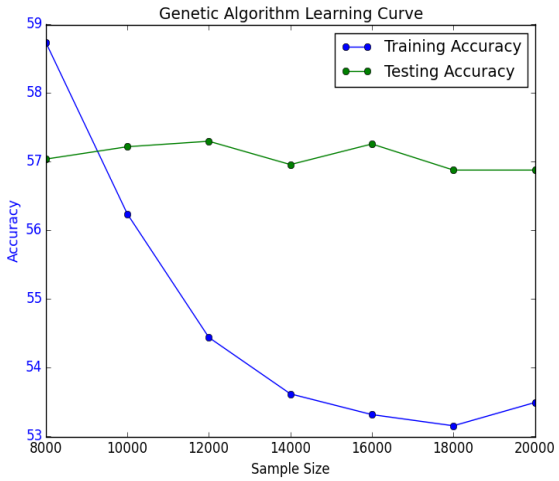


We can see from the above graphs that the accuracy of the Neural Network did not increase linearly with the increase in the number of iterations. Another interesting thing to notice is that the lower number of iterations gave better test score than more number of iterations. Random Hill Optimization with less than 1000 iterations gave an accuracy score of 57% whereas increasing the number of iterations did not increase the accuracy of the network. Similarly, for Simulated Annealing optimizations the testing accuracy does not increase with the increase in number of iterations. Interestingly, for genetic algorithm the testing accuracy actually decreases with the increase in accuracy.

We can conclude that the increasing the number of iterations does not increase the testing accuracy however, the time it takes to run each simulation increases significantly with the number of iterations. Therefore, by keeping the number of iterations constant and changing the number of training data the following curves were obtained.







With random hill climbing methods, as the number of sample size is increased both the training and testing error decreases significantly. But, with increasing sample size we can see that the testing accuracy increases which means that the random optimization is able to find good weights. However, the weights for neural network are not as optimal as given by the back propagation algorithm. This could have been improved by getting the average score based on many runs instead of just using one single run. The drawback for this is that it is computationally expensive and takes a lot of time.

Simulated annealing optimization gives much better accuracy than the other two algorithms. The testing accuracy increases with the number of sample size as the bias of classifier decreases. It seems that providing even more data would have increased the accuracy much further. It could be explained by the fact that instead of only doing hill climbing it also does some hill descending. It gave the best accuracy of about 57.5%. It is better than other three algorithms but, not as good as the back propagation algorithm to train the neural network.

Genetic algorithm does not show much variation in the testing accuracy with the number of sample size. However, we can see that the training accuracy decreases much faster than the testing accuracy which seems that the network trained with the genetic algorithm is not learning the training set itself but, figuring out the data set. We can see that the training accuracy decreases with the sample size which means the data was over-fit and now it's an under-fit. However, even with the genetic algorithm the accuracy is only about 57% whereas with the back propagation algorithm the accuracy was about 73%.

The poor performance of the optimization algorithms could be explained by stochastic errors where the starting point of each algorithm is randomly selected. Other factors could have been the presence of local optima where the algorithms are getting stuck. Each of the algorithms seem to be fixed on the accuracy of around 57% which suggests that the weight that have been converged to are the same. The subpar performance of the Genetic Algorithm can be explained by The Fundamental Theorem of Genetic Algorithms which assumes an infinitely large population. In this experiment we had to fix the population size such that the algorithms complete in a finite amount of time and the lack of very large dataset also could have been a hurdle.

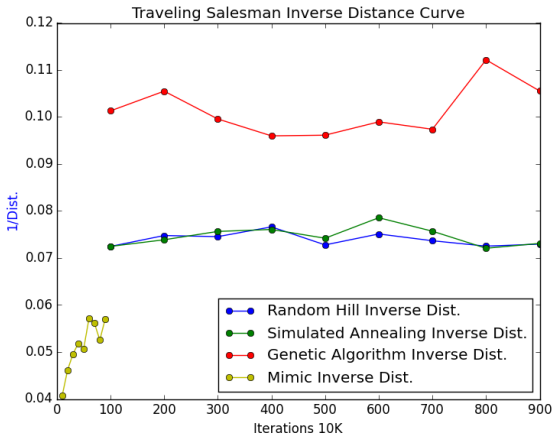
Traveling Salesman Problem

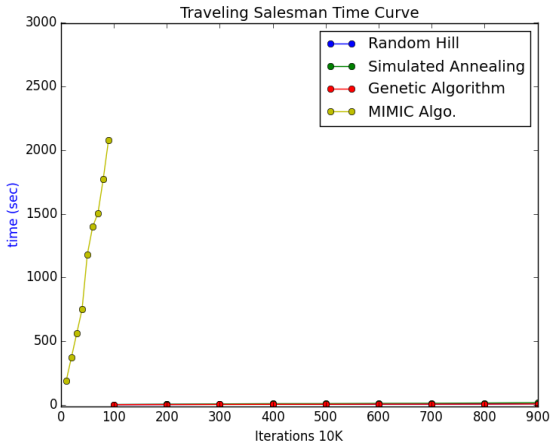
In the traveling salesman problem we are given a set of cities and the distance or weights between them. Our goal is to then find the minimum tour length to visit all the cities. Since this is a NP-hard problem there is no polynomial time algorithm to solve this in the general case. However, there are a number of heuristics which can be applied to problems under certain criteria like assuming the distances between the cities are symmetric. Given the cities {a,b} the distance dist(a,b) = dist(b,a) are the same going from one to the other.

Since, the problem is NP-hard the algorithms to find the optimal distance must have worst case running time that grows faster than polynomial. There are some programming techniques and heuristics to try and find the optimal structure however, these are not always efficient. Here, we analyze four randomization techniques Random Hill Climbing (RHC), Simulated Annealing (SA), Genetic Algorithm (GA) and MIMIC to find the optimal distance.

The goal of the Traveling Salesman problem is to minimize the overall distance travelled while visiting all the nodes. Even though we want to minimize the weights between the different cities or nodes in the graph. We can view this as a maximization problem if we take the inverse of the total distance to be the fitness function. Therefore, for this problem we analyze the graph and try and maximize the inverse of the round trip or the total distance. Hence our fitness function becomes 1/ (total dist.).

ABAGAIL was used to run each of the algorithms. In our experiment, the cities are just a list of randomly populated (x,y) points in a 2-D graph. The position of the nodes within the Euclidean space give it the weight. Our goal then is to minimize the distance travelled among all of them starting from a random node and iterating between the nodes. For each of the algorithms there is no stopping criteria instead we run the trainer for a fixed number of iterations. Hence, how each of these optimization techniques perform with the number of iterations is of importance. Below, we can see the performance of each heuristic with the number of iterations:





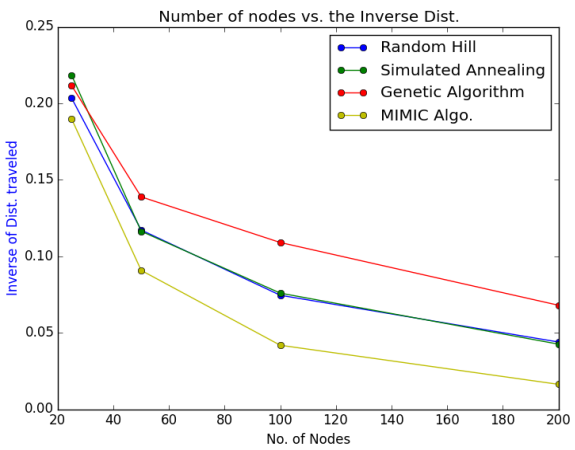
The inverse of the total distance is plotted on the graph for the graph of 100 nodes. Hence, higher the graph the better it is able to minimize the total distance travelled. Each of points are recorded by running the algorithms over the same graph and averaging the number of runs over 25 times.

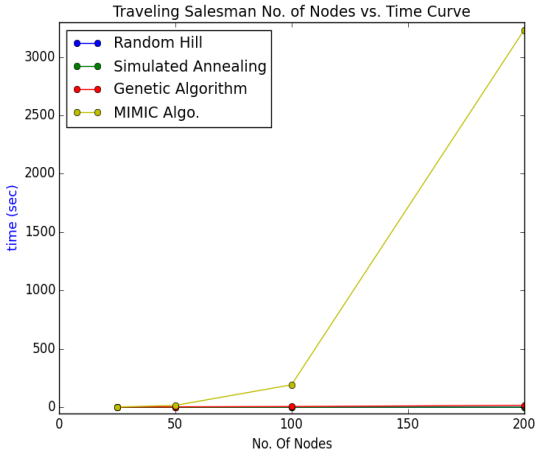
The iterations for three algorithms RHC, SA and GA are run 100K to 900K times except for the MIMIC algorithm which is only run for 100 times. But, for MIMIC each training iterations has multiple iterations within its training method. Which can be observed by the time it takes to run a single iteration of the MIMIC algorithm.

As can be observed in the graph above, Genetic Algorithm performs best in terms of minimizing the cost between 100 nodes in the graph. This can be attributed to the randomization of the algorithm which uses both crossover and mutation. For this particular problem the mutation rate is set to be at 20% and the initial population is 200 out of which 150 are chosen for crossover or ¾ of the total population.

Similarly, for RHC, only other variable that is manipulated is the number of iterations for training. For SA the initial temperature is set to be at 10^12 and cooled down at the rate of 0.95. MIMIC is initialized with 200 population and only half of it is kept from one population to other.

After running the algorithms for different iterations we can see that the number of iterations does not affect the score of the algorithm by a lot. However, MIMIC tends to be very expensive with increasing number of iterations. Hence, the number of iterations was fixed and the number of nodes in the graph was expanded to see how different algorithms behaved with bigger graphs or more cities. Below, we can see the plot for the time and the cost to visit each city or node in the graph.





As can be seen from the plot above, Genetic Algorithm is still better with bigger number of nodes or bigger graph. This can be attributed to the mutation and randomization in the graph where each of the nodes are chosen at random and visited in a random fashion. Thus, it is likely to find better fitness function. MIMIC performed the worst because the problem is not suited for MIMIC because the graphs do not have any apparent structure that needs to be stored and optimized.

However, we can see that Random Hill and Simulated Annealing performed similarly in terms of time and the fitness function. This can be attributed to the algorithm trying to only maximize the fitness curve. Different configuration of the nodes could lead to a better performance of the fitness function but, it might not be chosen by RHC and SA if the overall score is not maximized. Hence, they provide subpar performance to the genetic algorithm which has more randomization through crossover and mutation.

Four Peaks Problem

This is a problem to maximize the fitness function f(x) of an array of certain length. Suppose the string length is 100 fitness function is given by the following formula where T is the threshold:

z(x) = Number of contiguous Zeros ending in Position 100

o(x) = Number of contiguous Ones starting in Position 1

REWARD = 100 if (o(x) > T and z(x) > T) else 0

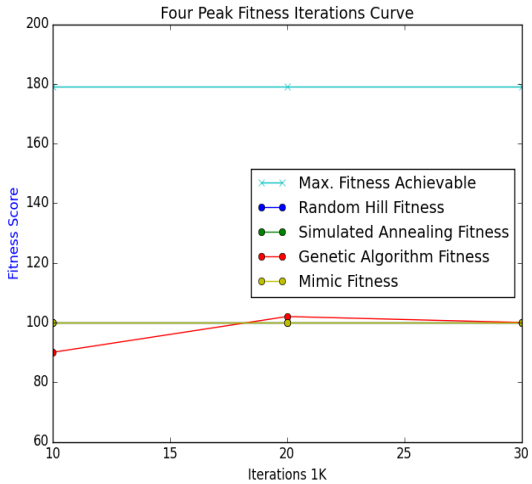
f(x) = MAX(z(x), o(x)) + REWARD

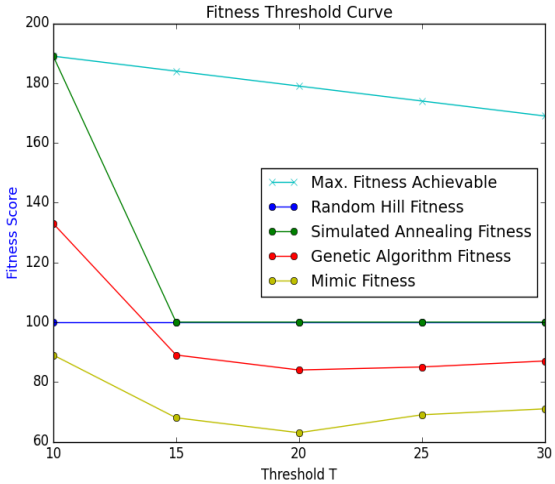
For T = 20, the fitness function f(x) can give the maximum value of 179 where either there are 21 contiguous zeros in the end and 79 ones in the beginning or there are 11 ones in the beginning and rest of the 79 are zeros. Either of these configuration give the maximum fitness value of 179.

This problem has been run in ABAGAIL, where the array is initialized to some random number '2' and the algorithm changes each position in the array to either 1 or 0. The problem is more interesting than the problem of counting number of ones in an array. For the count ones the fitness function is the count of 1's in the array, but, for four peaks the fitness function is assigned by getting the number of 0s or 1s above the threshold.

The problem is so named because there are four peaks or fitness score that can be achieved two of which are global optima and the other two are local optima. The value obtained by getting all 0's or all 1's for a 100 bit array is 100 which is the local optima. Whereas if the array has either 0 or 1 that is more than the threshold a reward is applied so for 100 bit array with the threshold of 20 the maximum fitness is 179. So, there are two configurations where the array can achieve max score of 169 and other two configuration where it can achieve score of exactly 100.

The optimization problem was run in ABAGAIL. Initially, the length of the array is kept at 100 and the threshold at 20. Number of iterations is changed to see how well the algorithms can optimize the given fitness function. The experiment is run over 25 times and the average cost and time are plotted below.





From the above graphs, it can be observed that all three algorithms RHC, SA, MIMIC are stuck in the local optima. Genetic algorithm does not seem to be stuck at the local optima but, it is not able to get the maximum achievable score. Even though the genetic algorithm is random from our experiments with higher number of iterations it seems to converge to the local optima. Random Hill optimization and Simulated Annealing algorithms tend to get stuck in local optima because the global optima is far away from the local optima. Even though SA is run with a high temperature of 10^11 it is not able to get away from the local optima and converges to the local optima perfectly.

We can also see the difference in fitness score by changing the threshold. Setting the threshold to be 10, 15, 20, 25, and 30 does effect the fitness score. We can see that for GA and MIMIC increasing the threshold decreases the overall fitness score because once the threshold reaches 50 for a 100 bit string the number of 0's or 1's have to be exactly half on both sides which is harder to achieve.

We can also observe that for Random Hill Climbing and Simulated Annealing the fitness score that is achieved is still 100 which is the local optima. The hill climbing algorithms are not suited for problems which have multiple local optima. Simulated Annealing should overcome this problem of local optima but, in this case the local optima is big enough that even SA cannot climb out of it.

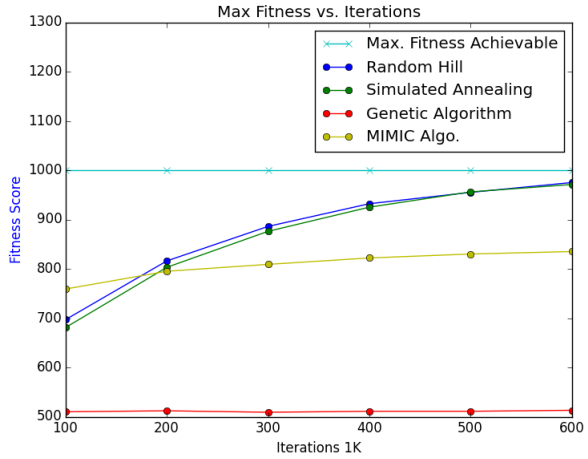
MIMIC is supposed to perform the best for these types of problems but, that was not observed in the experiment even though the problem was run for various iterations. MIMIC is supposed to maximize based on the configuration on both ends of the array. However, it seemed that MIMIC also got stuck in the local optima.

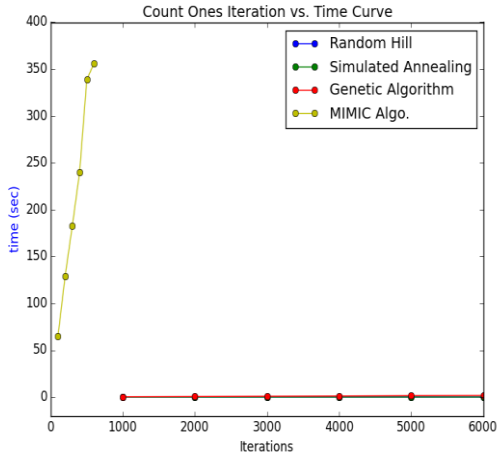
Count Ones Problem

This is a simple problem of maximizing the number of 1's in an array. Here, given an array of certain length like 100, 1000, or 2000 which are initialized to some random number '2' the optimization problem is to flip each position to 1 and count the number of 1's. The maximizing function is a simple function which counts the number of 1's in the array. Each of the optimization algorithm is supposed to optimize the fitness function of maximizing the number of 1's in the array.

This is a straightforward optimization problem where by each algorithm changes one of the position in the array randomly and chooses that array which has more number of 1's over the other one based on the fitness function. Since, the maximum fitness function is an array with all ones we can see how effective each algorithm is based on how close they are to the best score.

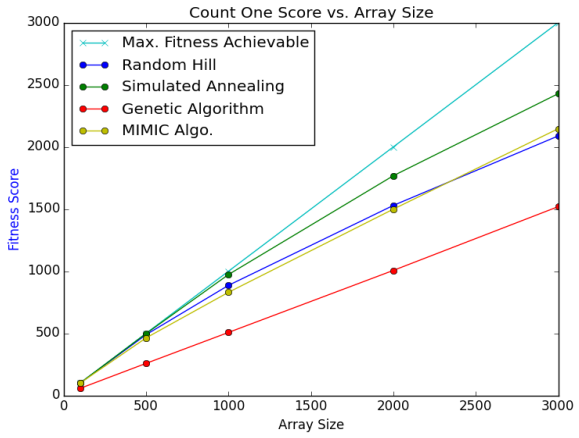
For each algorithm, Random Hill Climbing (RHC), Simulated Annealing (SA), Genetic Algorithm (GA) and MIMIC, we can analyze how well they perform based on the number of iterations provided for training. Given a higher number of iterations we expect the algorithm to better optimize the fitness function. The plot for each of the algorithm given the number of iterations and the maximum score achieved along with the time taken are plotted below.

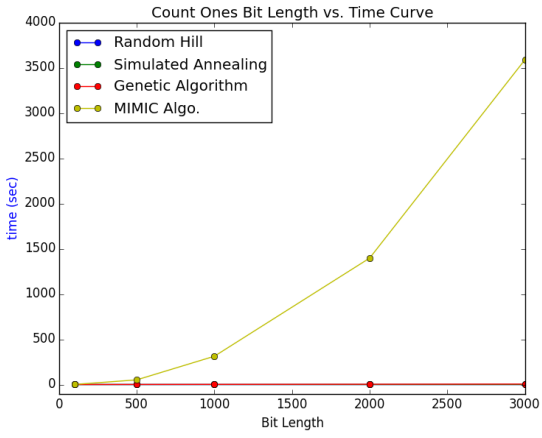




Each of the algorithms were run for an array of length 1000. Hence, the maximum achievable score is 1000. For each iteration it was run 25 times and the average score was taken in order to avoid error from randomness. From the plot above we can see that RHC and SA perform best given enough iterations. We can also see that given enough iterations the algorithm would actually converge to the maximum score. However, we can notice that the Genetic Algorithm does not perform well at all. It is because the GA has more randomness introduced due to crossover and mutation. MIMIC performs better than GA because the MIMIC algorithm takes the structure of the solutions into consideration and this problem has a structure where more number of ones give it a better fitness score.

The time taken for each of the algorithm has also been plotted which is similar to other problems where running time for MIMIC is much higher than other algorithms. After the initial analysis with the number of iterations we can also analyze the fitness score based on the size of the input. How do the algorithms perform given various sizes of the array? For these experiments the number of iterations are kept constant and only the size of the array is changed. The results for the experiments are plotted below:





From the above plots we can see that Simulated Annealing algorithm performs best given the various size of the array. Even though, it does not achieve the perfect score it does give the best overall score over varying array length from 100 to 3000. For smaller array it can be seen that RHC, SA, and MIMIC

perform the same whereas GA does not perform as well as other algorithms. For larger arrays RHC and MIMIC give similar fitness score whereas SA gives overall better performance.

Even, in terms of time RHC, SA, GA perform similarly whereas MIMIC dominates the curve for input arrays of bigger size. It can be attributed to the structure that MIMIC needs to maintain to optimize the cost function. Genetic algorithm would randomize the population even though the problem is to simply maximize the fitness function in a linear fashion by increasing the number of ones.

This problem is better suited for Random Hill Climbing and Simulated Annealing where the aim is to find the next neighbor which can maximize the fitness function which can be done in an uphill climb. In this case, the neighbor would be an array with more number of ones hence always finding the next best number gives the better answer. Therefore, Simulated Annealing performs much better than other algorithms and converges to the maximum score faster as it also randomly takes a downhill neighbor which could lead to a global optimization.

References:

Baluja, Shumeet, and Rich Caruana. "Removing the genetics from the standard genetic algorithm." *Machine Learning: Proceedings of the Twelfth International Conference*. 1995.

Isbell Jr, Charles L. "Randomized Local Search as Successive Estimation of Probability Densities." *A longer tutorial version of the 1997 paper on MIMIC that includes a derivation for MIMIC with trees. Can be accessed at http://www. cc. gatech. edu/∼ isbell/tutorials/mimic-tutorial. pdf*.

Chen, Luonan, and Kazuyuki Aihara. "Chaotic simulated annealing by a neural network model with transient chaos." *Neural networks* 8.6 (1995): 915-930.

Caruana, Rich, and Alexandru Niculescu-Mizil. "Data mining in metric space: an empirical analysis of supervised learning performance criteria." Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2004.

Online News Popularity. (n.d.). Retrieved September 21, 2015. http://archive.ics.uci.edu/ml/datasets/Online+News+Popularity