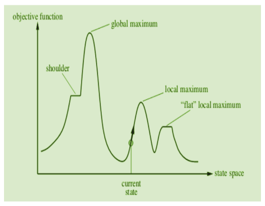
Part 1: Four Optimization Algorithms comparisons using three Problems

Before we start diving into each of the problems, let's understand each of the algorithms below.

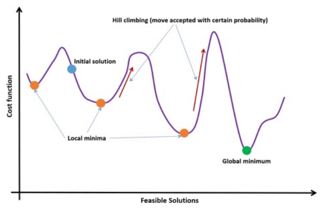
1. ### Randomized hill climbing

   Hill climbing is an optimization technique that belongs to the family of local search. The algorithm has an iterative nature, it starts with an arbitrary solution and then tries to find the better solution incrementally until there is no room for improvement. Hill climbing can find optimal solutions for convex problems whereas for other problems it can only find local optima which may or may not be the global optimum. The main advantage of Hill climbing is that it is very fast and doesn't not use too much memory.



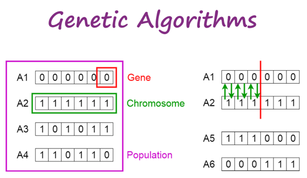
2. ### Simulated annealing

   Simulated annealing (**SA**) is an algorithm which in real life can be linked to Physical annealing. Physical annealing is a process wherein the metal is heated till the annealing point and then is cooled down slowly and converted into a hard structure. The process is repeated until the desired structure is achieved. Simulated annealing is built on similar concept of physical annealing but it's used for optimizing parameters. The algorithm is used in situations when there is a lot of local optimum and finding global optimum is preferred, general gradient descent might get stuck in these situations.



3. ### Genetic algorithm

   Genetic algorithm is a random-based classical evolutionary algorithm, basically random changes are applied to the current one in order to generate a new one. The algorithm is based on natural selection which reflects the process of biological evolution. At each step, the algorithm selects people at random who could be parents and then uses it to determine the children for the next generation.



   The genetic algorithm has 3 main steps.

   Selection rules: select parents which will contribute to the next generation population

   Crossover rules: combine two parents to form next generation children.

   Mutation rules: apply random changes in the child formation process.

4. ### MIMIC

   MIMIC is a novel optimization algorithm which is more reliable than several other algorithms. It converges faster. MIMIC begins by generating a random population of candidates chosen uniformly from the input space which has the likelihood of containing the optima. It then uses an effective density estimator to discover a common underlying structure about optima from the input space.

Let’s now dive into the randomized optimization problems and observe the behavior of each of the above-mentioned algorithms.

1. **Four Peaks Problem**

This is a simple toy problem whose fitness function works by counting the number of consecutive 0s at the start and no of consecutive 1s at the end of the string with an intent to return the maximum. In the scenario where no of 0s and no of 1s is above a threshold of T, the fitness function gets a bonus of 100. Thinking differently, there is a small peak with lots of 0s and a small peak with lots of 1s and then two large peaks with the bonus included in it. Since it has 4 peaks, it’s called the 4 peaks problem. This problem is Genetic Algorithm friendly.
Given an N-dimensional bitstring x, four peaks evaluation function is:

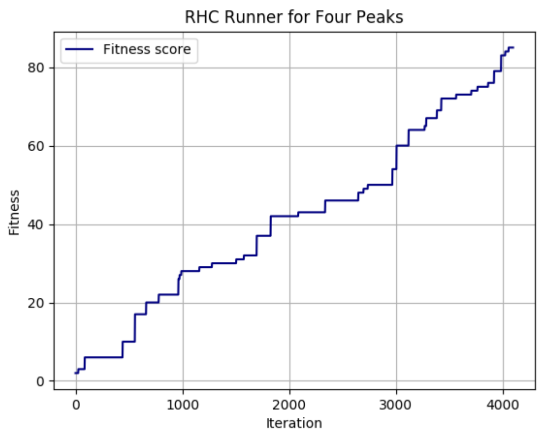
$$
\begin{aligned}
f(\mathbf{x}, T) &= \max\{tail(0,\mathbf{x}), head(1,\mathbf{x})\} + R(\mathbf{x}, T) \\
&\text{where} \\
tail(0,\mathbf{x}) &= \text{number of trailing 0's in } \mathbf{x} \\
head(1,\mathbf{x}) &= \text{number of leading 1's in } \mathbf{x} \\
R(\mathbf{x}, T) &= \begin{cases} N, & \text{if } tail(0,\mathbf{x}) > T \text{ and } head(1,\mathbf{x}) > T \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

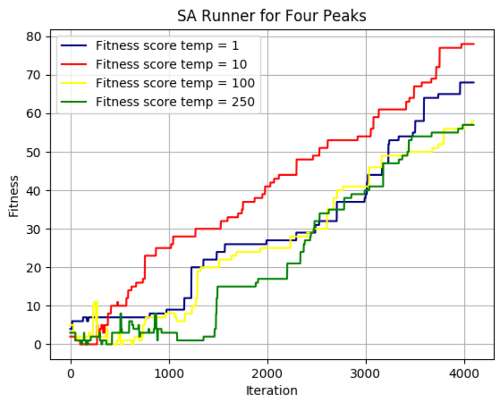
Suppose N= 100 and T = 10, the optimal fitness of 189 is achieved when the string has 89 1s followed by 11 0s or when the string has 89 0s followed by 11 1s. Thus, the function has two global maximas, one when the string has T+ 1 0s preceded by all 1s or T+ 1 1s preceded by all 0s. One thing to note here is that strings with no of 0s and no of 1s greater than T but having sub-optimal lengths can reach either of the global maximas by repeatedly flipping bits of largest of 0s or 1s at the end of each run. Ex : when Os = 15 and 1s = 50, hill climbing can reach the peak by flipping the bits at 51s position and then 52nd and so on until it reaches the 89 1s and 11 0s. This problem also has two suboptimal local optima having fitness of 100. One when there are 100 0s and 0 1s, second when the string has 100 1s and 0 0s. Hill climbing and Simulated Annealing will quickly get trapped in these local optimas. Ex : For a N = 100, consider the tail(0,x) = 25 and head (1,x) = 5, Random hill and simulated annealing will continue increasing the value of the tail(0,x) until it reaches the value of 100.The only way a string that has both tail(0,x) ≤T and head(1,x) ≤T can find global optima by hillclimbing and SA algorithm is if it continues to flip the bits at the shorter end despite the fact that fitness is not improving. This means continuously and repeatedly making right decisions while searching for large plateaus. This is not in practice as RCH will update the parameters only when fitness is improving. We can overcome this problem using Genetic Algorithm through crossover. Consider a population of string, some of which having head(1,x) >> tail(0,x) and some of which having tail(0,x) >> head(1,x) but none of them head(1,x) or tail(0,x) > T, crossover on such a population string will occasionally create individuals with head(1,x) > T and tail(0,x) > T and whenever this happens the child will receive bonus reward would be 100 and fitness value of child would be higher than the fitness value of parents.

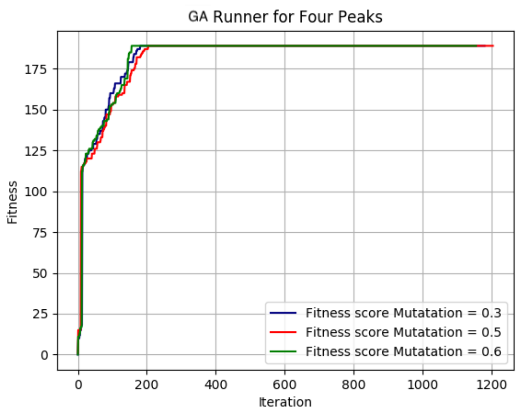
Crossover on a population of strings, some of which have head(1, x) >> tail(0, x) and others which have tail(0, x) >> head(1, x) but none of them with head(1, x) > T and tail(0, x) > T, is likely to create individuals with head(1, x) > T and tail(0, x) > T. When this happens, the child string will receive the additional reward of N and will have a higher fitness than its parents. Thus the four peak problem performs better on GA through crossover.

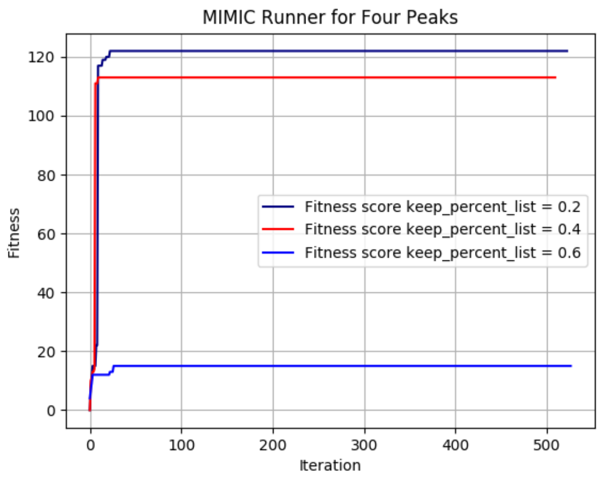
Now let's look at the results of the experiments and validate our concept above. For this experiment I use mlrose python library. All the 4 algorithms were run on the 4 peak problems and for a fair comparison, each of the algorithms was run for 4096 iterations. For GA the population was set to 200 and the experiments were performed on mutation rates of 0.3, 0.5 and 0.6. For SA experiments were performed on temperature values of 1, 10, 100 and 250. For MIMIC, the population value was set to 200 and experiments were performed on keep percent values of 0.2, 0.4 and 0.6.

Below are the results of the experiments with different parameters for each of the runs. We can see that for SA the best fitness score we see is at temperature value = 10, for GA we get the best fitness at mutation value of 0.6 as it is achieved in shortest no of iterations but more or less all of them achieve the optimal score of 189 in shortest number of iterations. For MIMIC the max score came to be 113 and was achieved on a 0.2 value of keep percent.

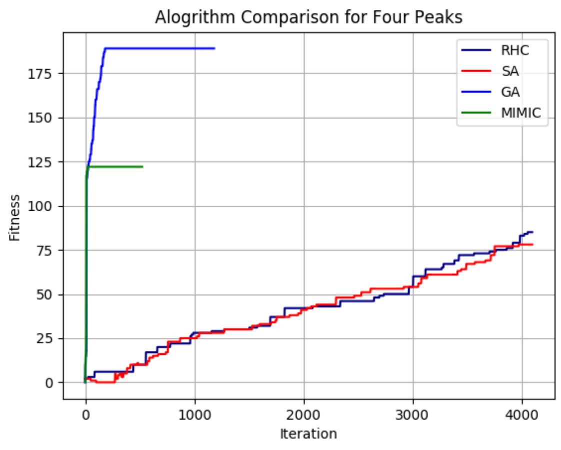


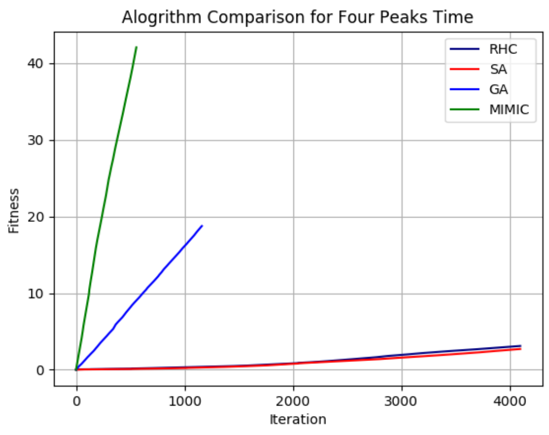






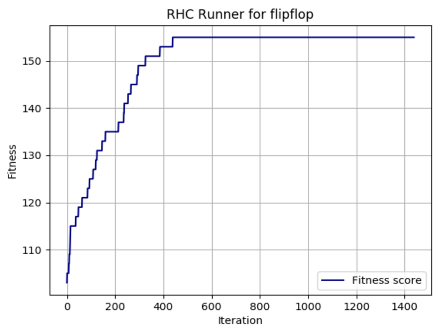
Below is the Comparison of best results of each of the above algorithms. The first graph is

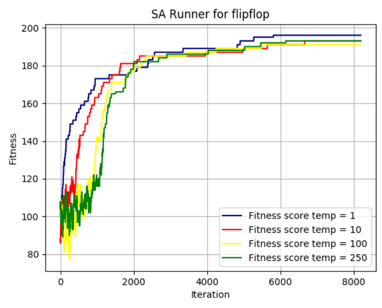


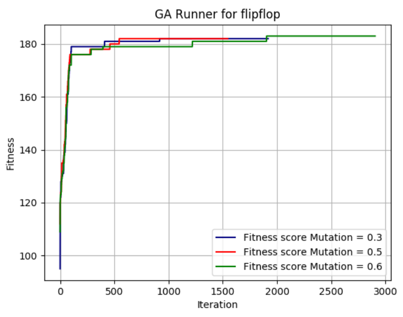


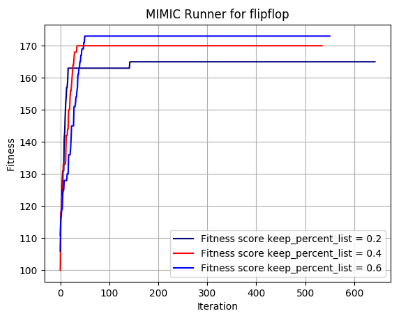
fitness score plotted over no of iterations and second graph is time plotted over no of iterations. We can clearly see that GA performed better compared to all of the algorithms achieving the best fitness score of 189 whereas other algorithms could not. We had the same expectation based on the problem statement above. Comparing time taken to achieve the best fitness score we can see that GA ran longer than SA and RHC, demonstrating that it is slower than RHC and SA.
Conclusion : GA was the best algorithm for the 4 peaks problem. RHC and SA were the worst in terms of fitness. Also for MIMIC, selecting the right parameter is very important as the fitness can drastically vary

2. **Flip Flop Problem:** Flip Flop problem counts the number of times the bit value is flipped in a bit string. In other words, if the current bit is 0 and the next bit is 1 it will count it as the next bit flipped. The max fitness in a bit string would be achieved in a case when all the bits in string are alternate to each other. This problem is a good candidate for SA as you could have a lot of local bitstring optimas in the bitstring and finding the best global optima is the main intent.Now let’s look at the results of the experiments and validate our concept above. For this experiment I use mlrose python library. All the 4 algorithms were run on the flip flop problem and for a fair comparison, each of the algorithms was run for 8192 iterations. For GA the population was set to 200 and the experiments were performed on mutation rates of 0.3, 0.5 and 0.6. For SA experiments were performed on temperature values of 1, 10, 100 and 250. For MIMIC, the population value was set to 200 and experiments were performed on keep percent values of 0.2, 0.4 and 0.6.
Below are the results of the experiments with different parameters for each of the runs. We can see that for SA the best fitness score came at temperature value of 1 but more of less all of temperature value shows similar results. For GA we get max fitness at 0.6 and for MIMIC at 0.6

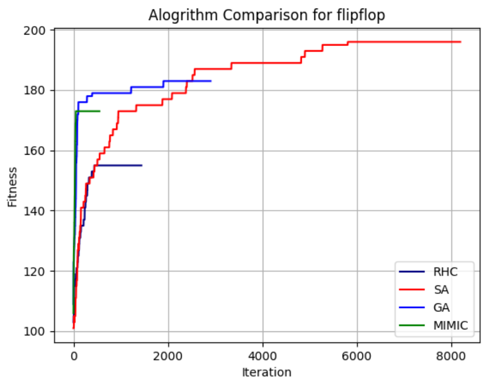


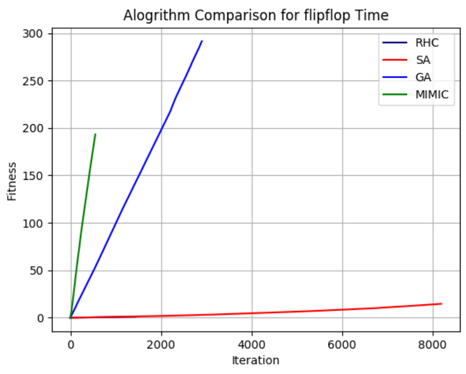






Below is the Comparison of best results of each of the above algorithms. The first graph is

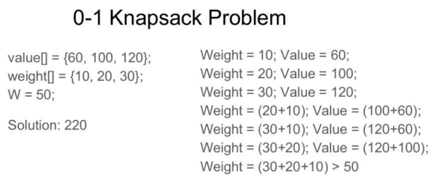




fitness score plotted over no of iterations and second is time over no of iterations.We can clearly see that SA performed better compared to all of the algorithms achieving the best fitness score of 193 whereas other algorithms could not. We had the same expectation based on the problem statement above. Comparing time taken to achieve the best fitness score we can see that GA ran longer than SA and RHC, demonstrating that it is slower than RHC and SA.

Conclusion : SA was the best algorithm for the flip flop problem. RHC, GA and MIMIC didn't achieve the best fitness and ended at early iterations.

3. **Knapsack Problem:** The knapsack problem is a common world problem of packing the most useful items without overloading the luggage. For a given set of items having a weight and value, the goal is to determine the number of each item that we can include in a Knapsack so that the total weight is less than the limit(W) and the total value of the collection is the maximum. This is a Non-Deterministic Polynomial time problem and runs in exponential time. The way to solve this problem is using a dynamic programming approach, basically first solve for a smaller set of knapsacks problems and then move to larger problems. In my mind, the best algorithm to solve this would be a MIMIC algorithm based on how it works. Given a list of items with weights (w) and value(v), the algorithm will randomly select some items and try to fit them in, it will also calculate the density value for those items. If it didn't reach the optimal fitness, the density value from the previous iteration would be used to select the next input sample and then fit in until we reach the optimal fitness. The process would be repeated till we reach the optimal value. In other words the self learning concept of


0-1 Knapsack Problem

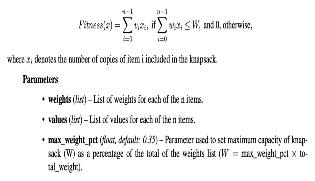
value[] = {60, 100, 120};
weight[] = {10, 20, 30};
W = 50;

Solution: 220

Weight = 10; Value = 60;
Weight = 20; Value = 100;
Weight = 30; Value = 120;
Weight = (20+10); Value = (100+60);
Weight = (30+10); Value = (120+60);
Weight = (30+20); Value = (120+100);
Weight = (30+20+10) > 50

MIMIC based on previous iterations will help solve this problem in an optimal way. We can also solve this using GA by selecting a list of items and then summing their values and checking if it fits in the knapsack if not then subtract twice the amount by which it is big and then try to fit in the knapsack. The fitness will get better if we select strings that are fit than the rest of the population.

Now let's look at the results of the experiments and validate our concept above. For this experiment I used the mlrose python library function defined in this function. The pct value I used was 0.6 and input values and weights were chosen randomly of length 9. All the 4 algorithms were run on the knapsack problem and for a fair comparison, each of the algorithms was run for 64 iterations. For GA the population was set to 100 and the experiments were performed on mutation rates of 0.3, 0.5 and 0.6. For SA experiments were performed on temperature values of 1, 10, 100 and 250. For MIMIC, the population value was set to 200 and experiments were performed on keep percent values of 0.2, 0.4 and 0.6.



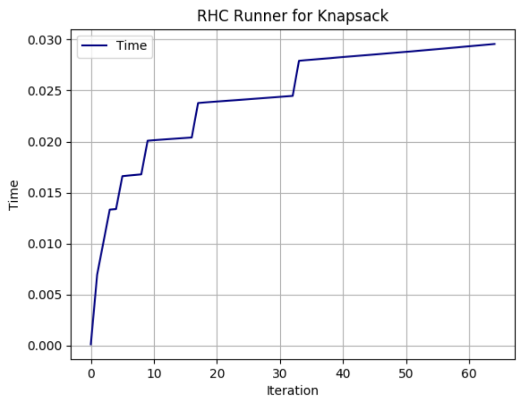
$$Fitness(x) = \sum_{i=0}^{n-1} v_i x_i, \text{ if} \sum_{i=0}^{n-1} w_i x_i \leq W, \text{ and 0, otherwise,}$$

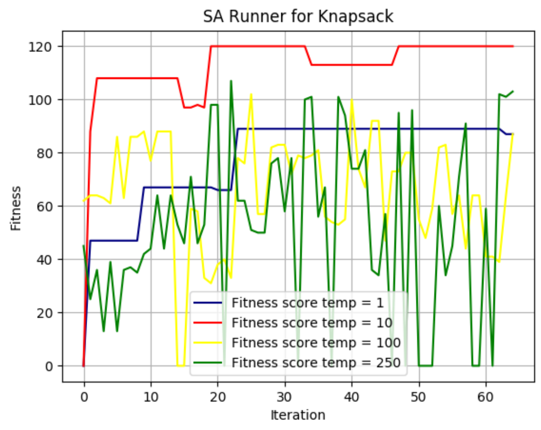
where x_i denotes the number of copies of item i included in the knapsack.

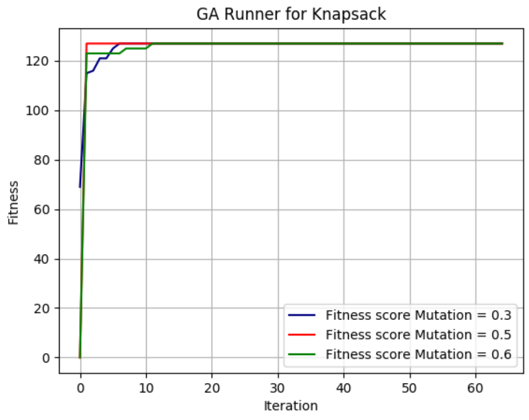
Parameters

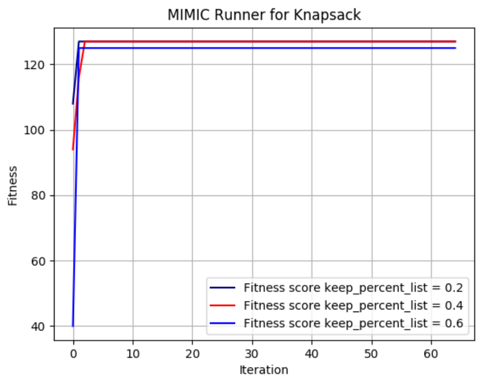
- **weights** (*list*) – List of weights for each of the n items.
- **values** (*list*) – List of values for each of the n items.
- **max_weight_pct** (*float, default: 0.35*) – Parameter used to set maximum capacity of knapsack (W) as a percentage of the total of the weights list (W = max_weight_pct × total_weight).

Below are the results of the experiments with different parameters for each of the runs. We can see that for SA the best fitness score we see is at temperature value = 10, for GA we get the best fitness at mutation value of 0.5 as it is achieved in shortest no of iterations but more or less all of them achieve the optimal score of 127 in shortest number of iterations. For MIMIC, the max score came to be 127 and was achieved at a 0.2 value of keep percent.

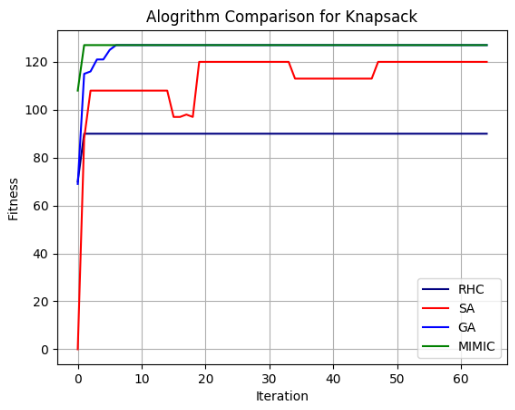


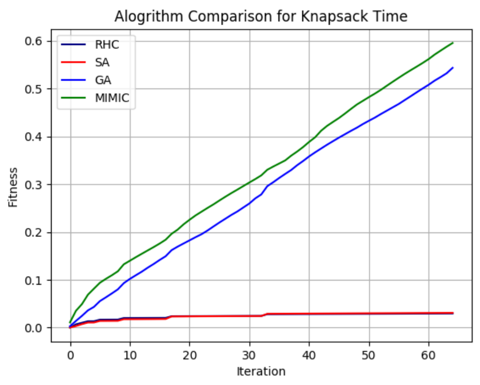






Below is the Comparison of best results of each of the above algorithms. The first graph is





fitness score plotted over no of iterations and second graph is time plotted over no of iterations. We can clearly see that MIMIC performed better compared to all of the algorithms achieving the best fitness score of 127 in a minimum number of iterations. Both MIMIC and GA achieved the optimal fitness score but GA took more iterations compared to MIMIC, We had the same expectation based on the problem statement above. Comparing time taken to achieve the best fitness score we can see that GA and MIMIC ran longer than SA and RHC, demonstrating that both of these algorithms are slower than RHC and SA.

Conclusion : MIMIC was the best algorithm for the knapsack problem. RHC and SA were the worst in terms of fitness. MIMIC iterates slower than the other algorithms, however it generates the best fitness score in this case.

Part 2: Neural Network Weight Optimization

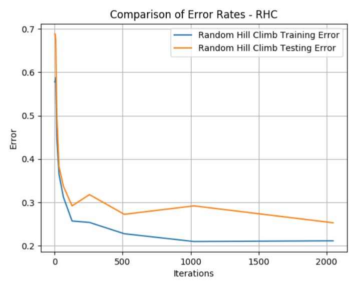
In any Machine Learning Models, the goal is always to fit the model using the parameter values which minimizes the amount of error. The main intent of this experiment is to find the optimal weight values of the Neural Network Model. To achieve the optimal weights, we will use the 3 randomized optimization algorithms.

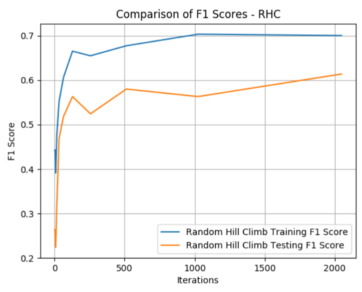
First Step is the dataset selection: I used the same dataset(Women Diabetes Prediction), the one I used in my previous Supervised Learning Neural Network Model with back propagation. The dataset contains the features of women diabetes records with an intent to predict if a women would be diabetic or not. The dataset seems balanced with 34.89% as Yes and 65.11.% as No and no of records for Yes and No are 268 and 500 respectively. The data was partitioned in proportion of 80:20 for training and test purposes.

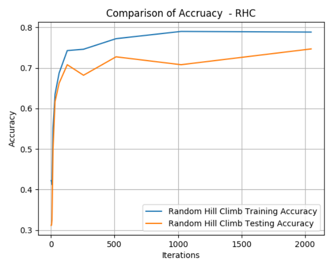
Getting Ready for Experiment : For this experiment I used the NeuralNetwork() class of mlrose python library. The class contains a fit function which internally implements all the steps needed to solve an optimization problem which are defining a fitness function, defining the problem and then deciding on the algorithm to be used for optimization. The fit function would determine the optimal weights but that is not my end goal here, my goal is to determine these optimal weights so that I can use the fitted model and then do future predictions on new samples of data and this is exactly what I did in my experiments which is demonstrated below. Once I finished fitting my model I used the same initialized object to predict my test data and then determined the effectiveness of the model using performance metrics like accuracy, loss and F1 scores. The Neural network setup has 30 nodes,has learning rate of 1 and uses relu activation. The Model was run with random hill, simulated algorithm and Generic Algorithm over these iterations [1,2,4,8,16,32,64,128, 256,512,1024,2048] and the accuracy, error and F1 score was plotted over different iterations. I also compared the different algorithms against each other.

Experiment results:

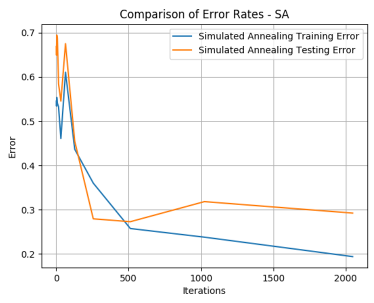
Random hill Climbing: Below are the charts for RHC. Looking at the F1 score graph, I noticed that the train and test model follows the same convergence path. I also noticed the accuracy goes up as we increase the number of iterations and the error goes down and converges towards a 0 value. I ran this for multiple times and noticed that the loss is not same in each run, it varies and there are cases when the loss is quite high, one reason I can think of is that the RCH got stuck in local optima. The test accuracy came to be 74%.

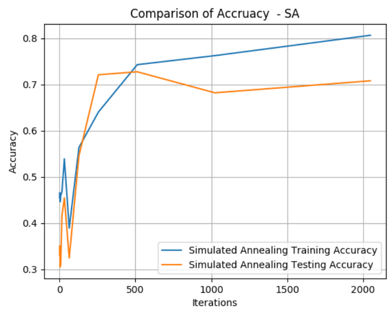


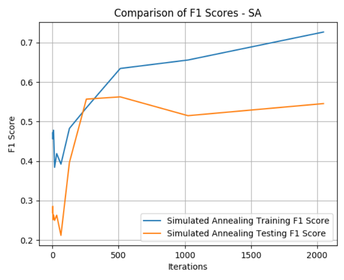




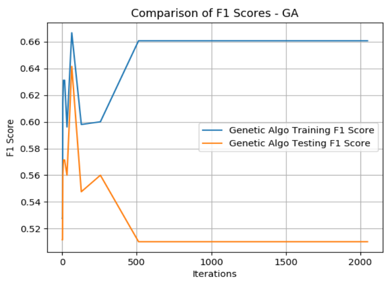
Simulated Annealing: Below are the charts for SA. Looking at the loss curve, I noticed that at high values of temperature or in the early iterations, SA has lots of fluctuations which means it can accept local bad moves at high temperature values and as the temperature value reduces with increase in number of iterations the training and test error converges on the same path and graph gets smoother. Looking at the F1 score, I noticed that the model accuracy gets better later when the temperature reduces and the chances of it getting stuck in the local optima is negligible. We can see that the train and test F1 scores later converges towards each other. The test accuracy came to be 70%

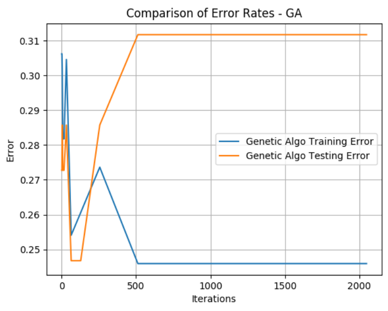


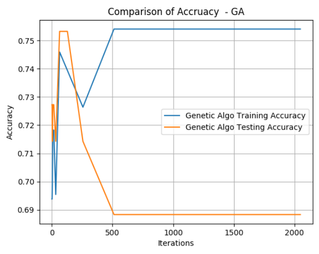




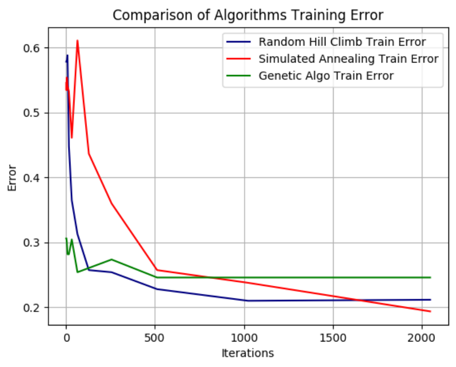
Genetic Algorithm : Below are the charts for GA. Looking at the loss curve, there is a lot of fluctuations between training and test error. We can see that in early iterations both training and test error fluctuates a lot and then converges but test error is quite high compared to training error. This algorithm performed poorly on my dataset compared to other algorithm.I think this is due to the fact that the population size wasn't significant enough for it to perform better. One way to improve that would be to increase the population size and mutation but that would result in longer run times so I didn't test that. Looking at the accuracy and F1 score, we notice the same behavior, in early iterations it fluctuates a lot and then converges with reduced F1 test score and increased training F1 score. The test accuracy came to be 68%.

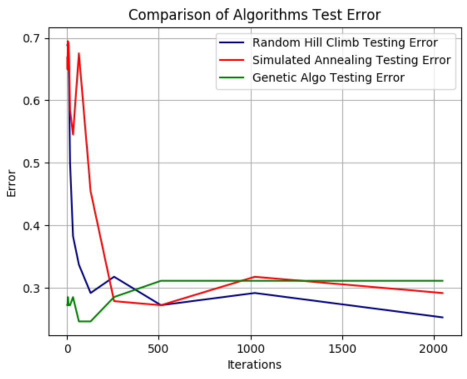


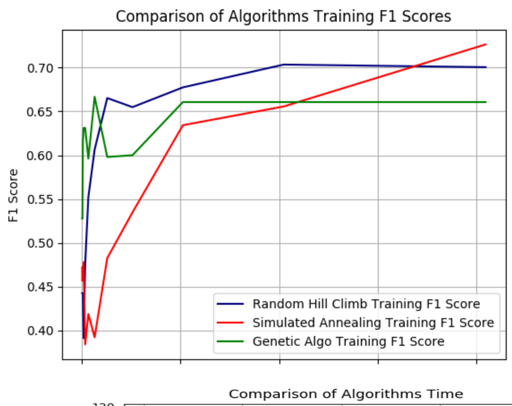


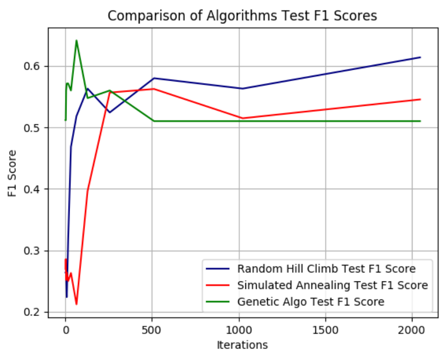


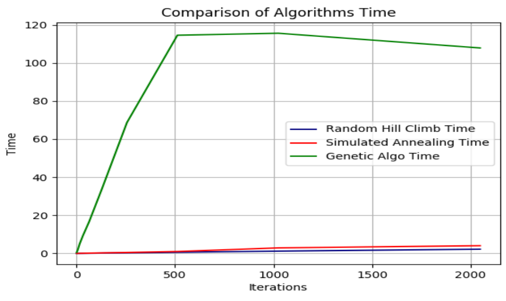
Comparison of different algorithms: To compare RHL, SA and GA algorithms, I plotted time, error, accuracy and F1 score for each of the algorithms together. Among all the algorithms RHC gives the best accuracy on my dataset and GA gives the worst accuracy. GA takes the most amount of time compared to other algorithms as depicted in the time comparison graph below. Looking at the test error, GA seems to have most of the error. Comparing my Neural Network with backpropagation I see that the accuracy for all the 3 optimization algorithms is higher than my bakprop's accuracy which was 67%. I have also displayed the stats on different performance parameters in below table.











Conclusion: All of the 3 randomized optimization algorithms performed compared to backpropagation and have accuracy and F1 score higher than backprop and error less than back prop. The fastest among the above is RHL and the slowest is GA. We can improve the performance of these algorithms by using cross validation to tune their hyperparameters which could be step size for RCH, decay rate for SA, population size, mutations for GA.

Algorithm	Train Error	Test Error	Accuracy	Test F1Score	Time
RHC	0.2117	0.2523	0.7465	0.61386	2.244
SA	0.1938	0.2922	0.7077	0.5454	4.055
GA	0.2459	0.3116	0.6883	0.5102	107.85

Based on the experiments above, here are some of the advantages/disadvantages of each of these algorithms.

- RHC
  - Very Fast and Suitable for Simple Problems
  - Can get stuck in local optima and may never reach global optima
- SA
  - Statistically guarantees optimal solution
  - Good for problems which has a lot of local minima
  - Doesn’t converge well at high temperatures
  - Parameter tuning is important for good results
- GA
  - Good for complicated problems.
  - Good for cases when crossover is needed.
  - Usually Slow Parameter tuning is important for good results
  - Very volatile at early iterations
- MIMIC
  - Good for problems which are complicated and greedy in nature and have exponentially time.
  - Very Slow.

References :

1. MIMIC - Jeremy S. De Bonet, Charles L. Isbell, Jr., Paul Viola, MIMIC: Finding Optima by Estimating, sourced from http://www.cc.gatech.edu/~isbell/tutorials/mimic-tutorial.pdf
2. Knapsack https://en.wikipedia.org/wiki/Knapsack_problem
3. https://medium.com/@duoduoyunnini/introduction-implementation-and-comparison-of-four-randomized-optimization-algorithms-fc4d96f9feea#_ftn1
4. https://www.ri.cmu.edu/pub_files/pub2/baluja_shumeet_1995_1/baluja_shumeet_1995_1.pdf
5. https://towardsdatascience.com/fitting-a-neural-network-using-randomized-optimization-in-python-71595de4ad2d
6. https://readthedocs.org/projects/mlrose/downloads/pdf/stable/
7. https://machinelearningmastery.com/how-to-calculate-precision-recall-f1-and-more-for-deep-learning-models/
8. https://mlrose.readthedocs.io/en/stable/source/decay.html
9. https://en.wikipedia.org/wiki/Four_Peaks