

## 1. INTRODUCTION

The purpose of this report is to evaluate 4 randomized optimizers on 3 problems namely four peaks, Traveling salesperson and Knapsack. The second part of the report evaluates the performance of a neural network for the 'Breast Cancer' dataset from my Assignment 1. This evaluation is done by comparing 3 randomized optimizers against the standard gradient descent optimizers usually used in neural networks.

## 2. PART 1

The present analysis is carried out using MLROSE implementation of randomized optimization algorithm library. In the first part, three problems are selected and analyzed using the 4 optimizations algorithms of Simulated annealing (SA), Genetic Algorithm (GA), Randomized hill climb (RHC) and Mutual-Information-Maximizing Input Clustering (MIMIC).

It is to be noted that the optimizers are run 3 times with different seeds each time with different initial state to ascertain that they are reaching the optimum.

As the analysis proceeds it goes on explaining the phenomenon like cross over, mutation, restarts, temperature and basically all the hyperparameters in the context of problem.

### Hyperparameter tuning

I ran a grid search CV over the following set of hyperparameters for the 3 problems: Four peaks, Knapsack and Travelling salesperson. A general configuration of the parameters is as follows (specific to each problem and its size the set of parameters differed from the following table):

N(Problem size)	Algorithm	Hyperparameter
40	SA	temperature_list=[1, 10, 50, 100, 250, 500, 1000, 2500, 5000, 10000]
40	GA	population_sizes=[200,400], mutation_rates=[0.1,0.3,0.5,0.7]
40	RHC	Restarts=[20,40]
40	MIMIC	population_sizes=[200,400,800,1600,3200](MIMIC converged to optimum only for population >=1600), keep_percent_list=[.1,0.5,0.75]

As the size of the problem increase, we include more parameters to set the optimizers' parameter to catch the complex optimal fitness function hyperplane.

### 1. Four peaks

This problem consists of two global maxima and two suboptimal local maxima with a basin of attraction. As the size of this problem increases the basin of attraction for the inferior local maxima becomes larger and the difficulty of optimization increases. The two sharp global optima with wide basin of attraction are expected to trap SA and RHC in contrast to GA and MIMIC. In the following analysis we discuss the performance of these algorithms.

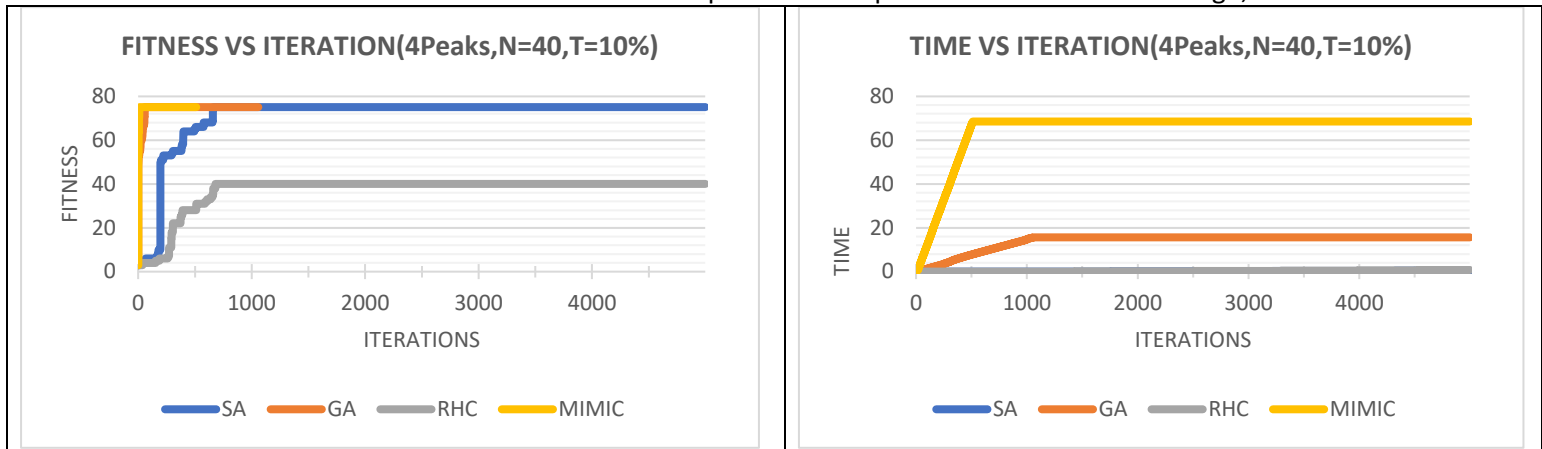


Figure: [4 peaks problem] (a)(Left) Fitness value curve, (b)(Right)Computation item curve

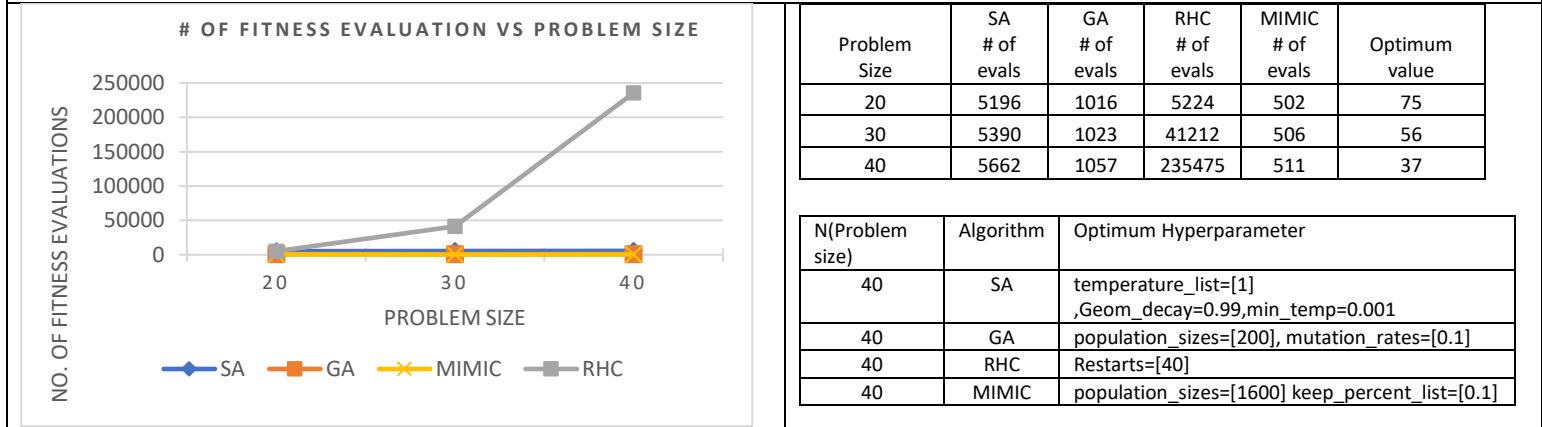


Figure: [4 peaks problem] (Left) # of Fitness evaluation curve, (Right)#of evaluations table

From the figure, MIMIC is the first to reach optimum followed by GA, SA and RHC (after many iterations and CPU time units not shown in graph). The 4 peaks problem can be mathematically stated as:

$f(\mathbf{x}, T) = \max\{\text{tail}(0, \mathbf{x}), \text{head}(1, \mathbf{x})\} + R(\mathbf{x}, T)$ <p>where</p> $\text{tail}(0, \mathbf{x}) = \text{number of trailing 0's in } \mathbf{x}$ $\text{head}(1, \mathbf{x}) = \text{number of leading 1's in } \mathbf{x}$ $R(\mathbf{x}, T) = \begin{cases} N, & \text{if } \text{tail}(0, \mathbf{x}) > T \text{ and } \text{head}(1, \mathbf{x}) > T \\ 0 & \text{otherwise} \end{cases}$	<p>For an N-dimensional bitstring <math>\mathbf{x}</math>, the global maxima is achieved when <math>T</math> (expressed as a % of <math>N</math>) value is such that there are <math>T+1</math> trailing 0's preceded by all 1's or there are <math>T+1</math> leading 1's followed by all 0's. In general, the global maximum can be given by <math>2N-T-1</math></p>
--	--

Let's assume a case of  $N=100$ ,  $\text{head}(1, \mathbf{x}) < T$  (say 8) and  $\text{tail}(0, \mathbf{x})=20$ . SA and RHC will increase the value of tail to 100. The only way to reach maxima is if the string keeps flipping bits at the shorter end even if the fitness is not improving. This can be very well captured by GA's crossover and MIMIC's exploration of the structure of problem by its information propagation mechanism. This explains their rapid convergence. RHC with a maximum restart of 5000 eventually captures such a case but at a much later stage and can be understood as a random coincidence. However, it is observed that SA was able to capture the maximum after around 5000 iteration in a much lesser time than all four algorithms (with an initial temperature of 1 itself). GA and MIMIC captured optimum after around 1000 and 500 iteration and the time taken was 68-time units and 15 units, respectively.

The figure also shows the number of fitness function evaluation vs problem size. It is observed that MIMIC outshines all other optimizers with the least number of fitness iterations for any problem size followed by GA, SA and RHC. This pattern is common for all the problems in this analysis. MIMIC's performance can be attributed for its ability to explore the underlying structure of the problem and propagate this information in successive steps to the population.

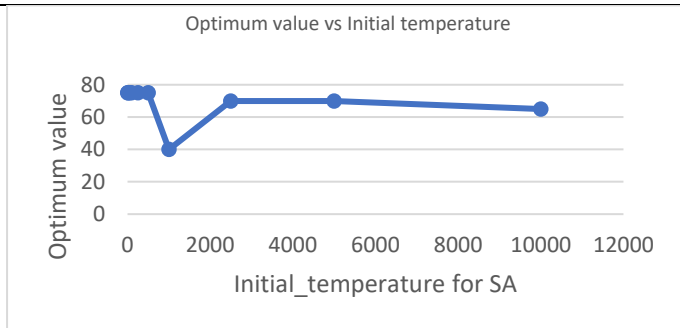
## Hyperparameter analysis

## GA

Population->	200	400
Mutation (below)		
0.1	75	75
0.3	75	75
0.5	75	75

It is observed that for any configuration of GA optimizer it reaches the global optimum. Thus, the obvious choice was to go for the simplest and least time-consuming config with population 200 and mutation 0.1. This observation also highlights that GA is best suited for this problem based on ease of convergence.

## SA



The optimizer reaches the global optimum of 75 if the initial\_temp was 1,10,50,100,250,500, however as initial temperature increase more the optimizer reaches a suboptimal value. It is expected as too high an initial temperature results in too much exploration without exploitation.

## MIMIC

Population->	400	800	1600	3200
keep_pct(below)				
0.1	66	71	75	75
0.5	54	57	64	53
0.75	49	53	55	59

2 factors are at play here. If we move horizontally for a given keep\_pct the optimum improves (increase) with population increase. The higher the population enables a better approximation of distribution in MIMIC. We want to keep a lesser percent at every time step providing a good approximation and lower kl divergence.

**RHC** : Optimum was reached within 40 restarts

## PERFORMANCE

	Based on iterations	Based on time	Based on combined time and iteration	Based on # of evaluations
Best Algorithm	MIMIC	SA	GA	MIMIC

## 2. TSP

This is an NP-hard problem. There is a salesman who must travel a set of cities for the purpose of marketing. However, he wants to travel the cities in a sequence such that the total distance travelled is least. The way I have set up these cities is as follows:

The cities are laid out in form of a grid of 5 X (variable number) =problem size. The first city is at (0,0). X coordinate range is 0-variable number and y coordinate range: 0 to 5.

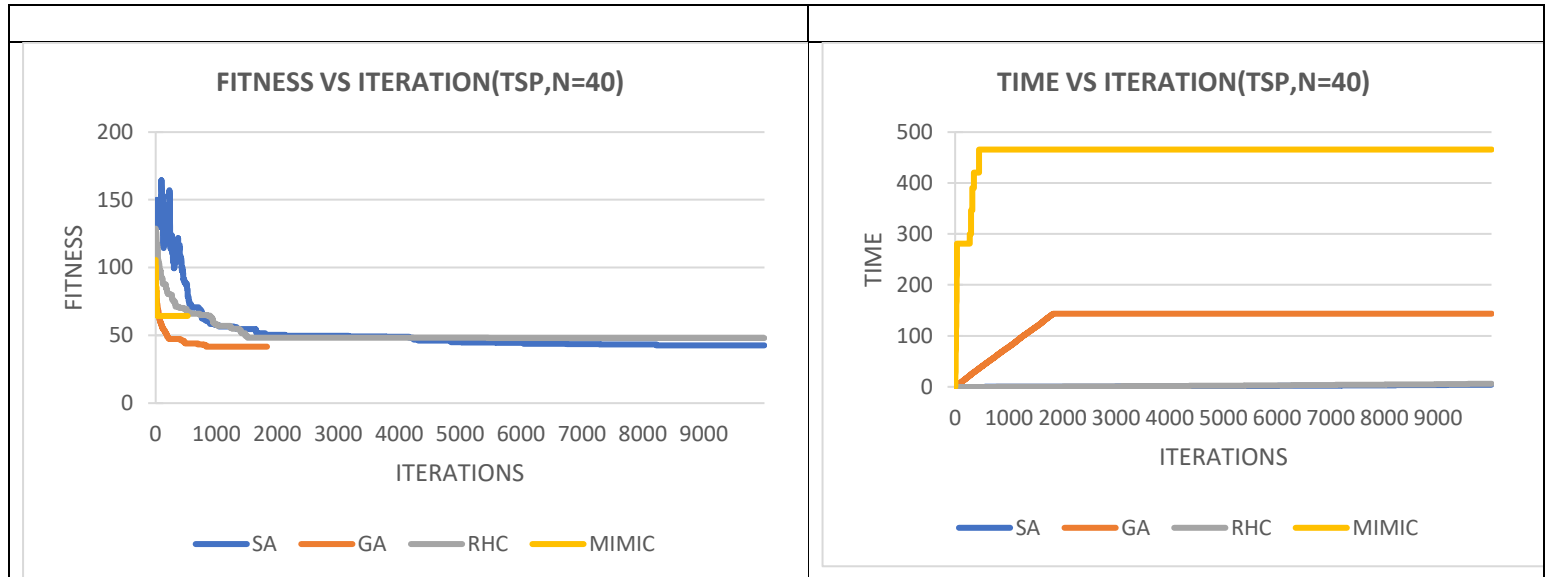


Figure: [4 peaks problem] (a)(Left) Fitness value curve, (b)(Right)Computation item curve

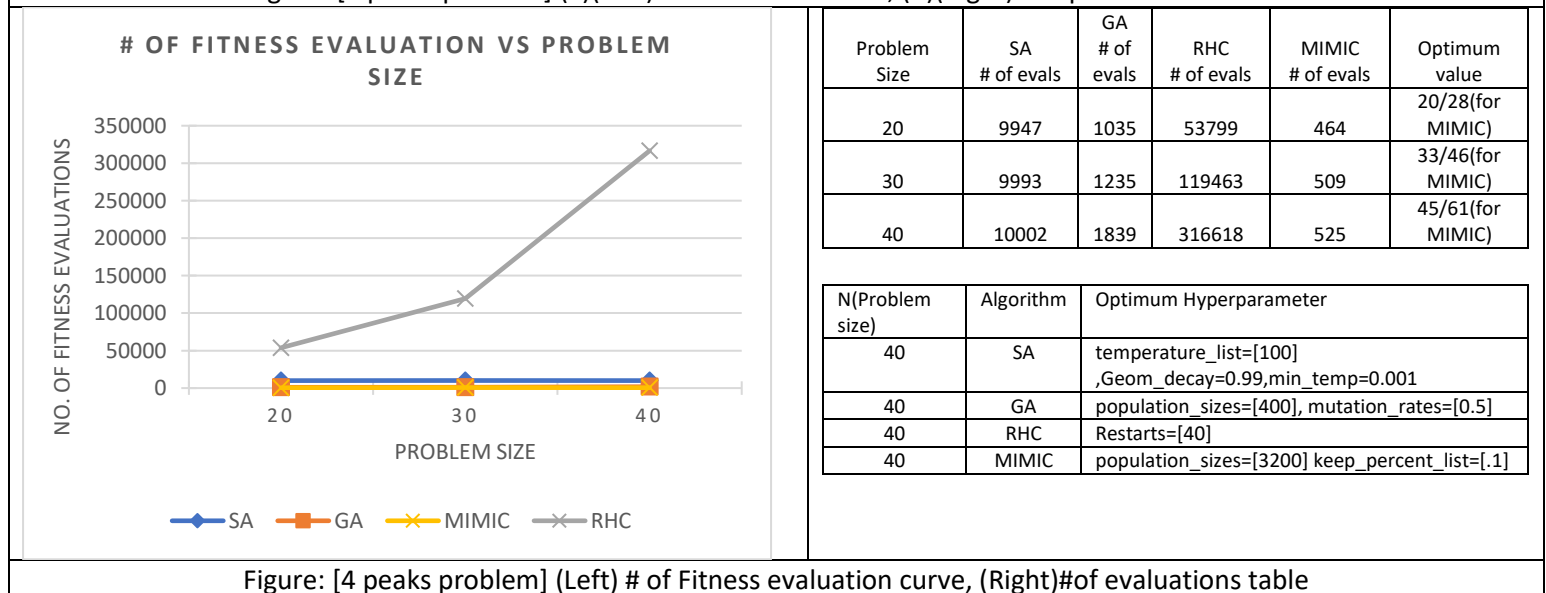


Figure: [4 peaks problem] (Left) # of Fitness evaluation curve, (Right)#of evaluations table

From the loss curve it is observed that MIMIC fails to reach the optimum values before it plateaus out at 61, while GA and SA reach it eventually and RHC reaches it after several iteration (300,000). MIMIC uses a joint probability to exploit the underlying structure and form dependencies within data. However, for TSP different optimum solutions which are very close in terms of their fitness function values (total distance travelled by salesman) may differ significantly in their underlying structure (the sequence of points travelled). Therefore, MIMIC fails to perform in this problem. GA on the other hand due to its ability to explore the parameter space thoroughly because of cross over is quick to converge, however it takes a lot of time for reasons explained earlier. With a starting temperature of 100, SA converges takes 10,000 evaluation to converge as compared to 1839 for GA, but the time taken by SA is 100 times lesser than GA. Thus, SA is the clear winner in this case. It explores the parameter

space quickly at high temperature (100 for problem size of 40) and converges as the temperature cools down. However, I believe that SA convergence would become more difficult as the problem size will increase. In such a case I believe a higher initial temperature will be needed. However, if the trade-off in total wall clock time beats other algorithm even with an increased problem size, SA shines out overall for the TSP problem.

### Hyperparameter Analysis

GA																		
Population->	200	400	<p>A higher population size of 400 with a higher mutation gives the best optimum value for GA. As explained above it follows from the way how GA optimization works.</p>															
Mutation (below)																		
0.1	46.23335	46.39835																
0.3	42	42.89																
0.5	43.89949	41.65																
SA																		
<p>Optimum value vs Initial temperature</p> <table><caption>Data points for Optimum value vs Initial temperature (approximate)</caption><thead><tr><th>Initial_temperature for SA</th><th>Optimum value</th></tr></thead><tbody><tr><td>0</td><td>51</td></tr><tr><td>500</td><td>43</td></tr><tr><td>1000</td><td>46</td></tr><tr><td>2500</td><td>46.5</td></tr><tr><td>5000</td><td>47</td></tr><tr><td>10000</td><td>46.5</td></tr></tbody></table>			Initial_temperature for SA	Optimum value	0	51	500	43	1000	46	2500	46.5	5000	47	10000	46.5	<p>100 provides sufficient exploration, however as initial temperature increases too much exploration does not provide as good optimum.</p>	
Initial_temperature for SA	Optimum value																	
0	51																	
500	43																	
1000	46																	
2500	46.5																	
5000	47																	
10000	46.5																	
MIMIC																		
Population->	400	800	1600	3200	<p>2 factors are at play here. If we move horizontally for a given keep_pct the optimum improves (decrease) with population increase. The higher the population the more details of structure are being captured, however with pop size=3200, MIMIC structure exploration cannot keep up as KL divergence becomes big.</p>													
keep_pct(below)																		
0.1	109.766	84.80847	68.58286	138.5601														
0.5	83.41849	80.21424	60.64082	115.8949														
0.75	87.81174	82	61.35541	197.5413														

### 3. Knapsack

This is an NP hard problem. The problem statement is as follows: “Given weights and values of  $n$  items, put these items in a knapsack of capacity  $W$  to get the maximum total value in the knapsack. In other words, given two integer arrays  $val[0..n-1]$  and  $wt[0..n-1]$  which represent values and weights associated with  $n$  items respectively. Also given an integer  $W$  which represents knapsack capacity, find out the maximum value subset of  $val[]$  such that sum of the weights of this subset is smaller than or equal to  $W$ . You cannot break an item, either pick the complete item or don't pick it (0-1 property)”. [1]

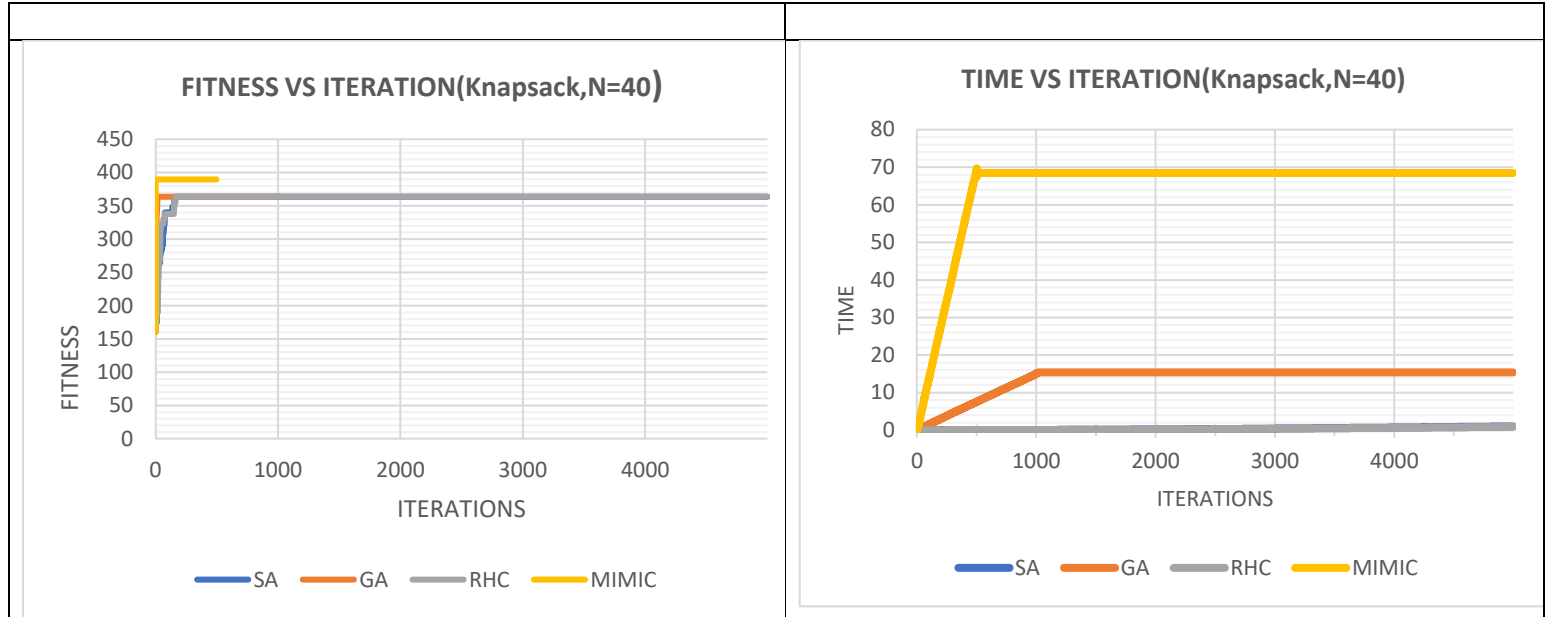


Figure: [4 peaks problem] (a)(Left) Fitness value curve, (b)(Right)Computation item curve

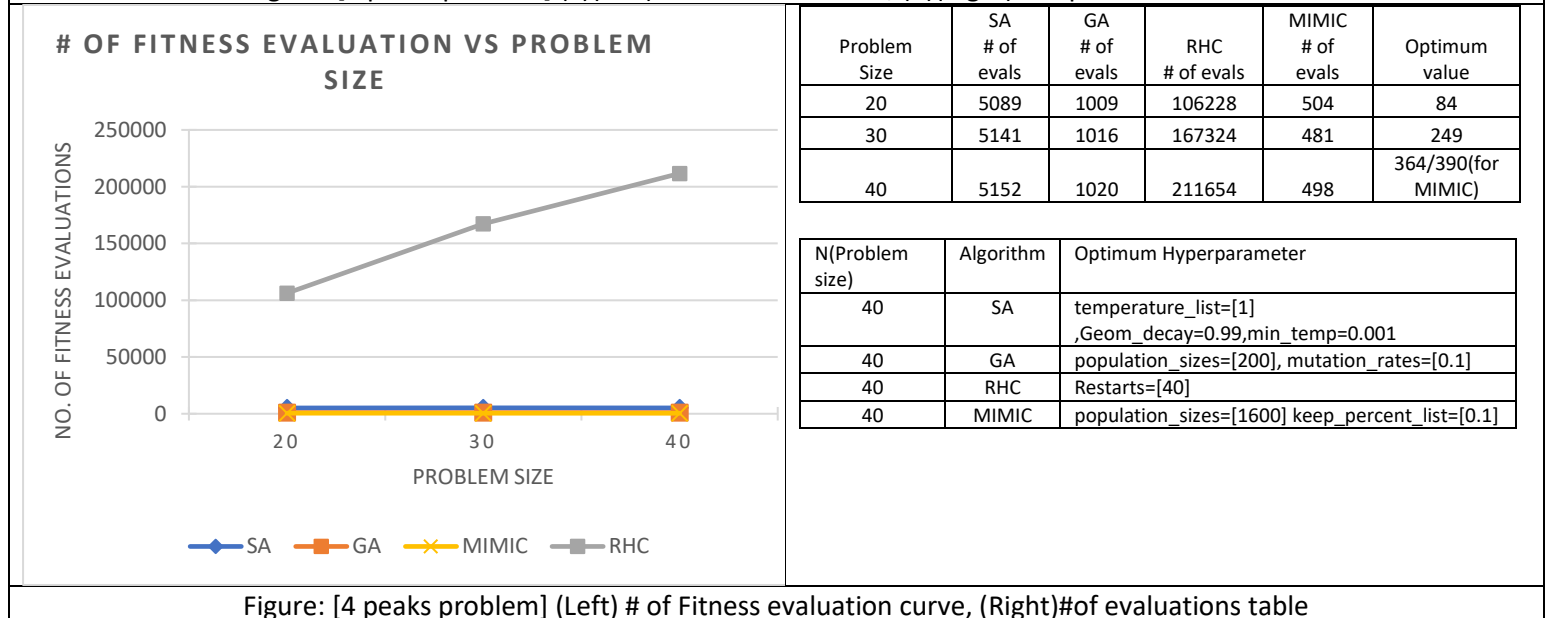
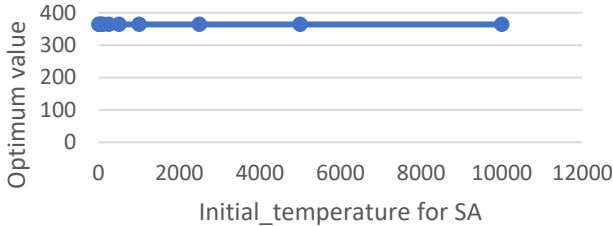


Figure: [4 peaks problem] (Left) # of Fitness evaluation curve, (Right)#of evaluations table

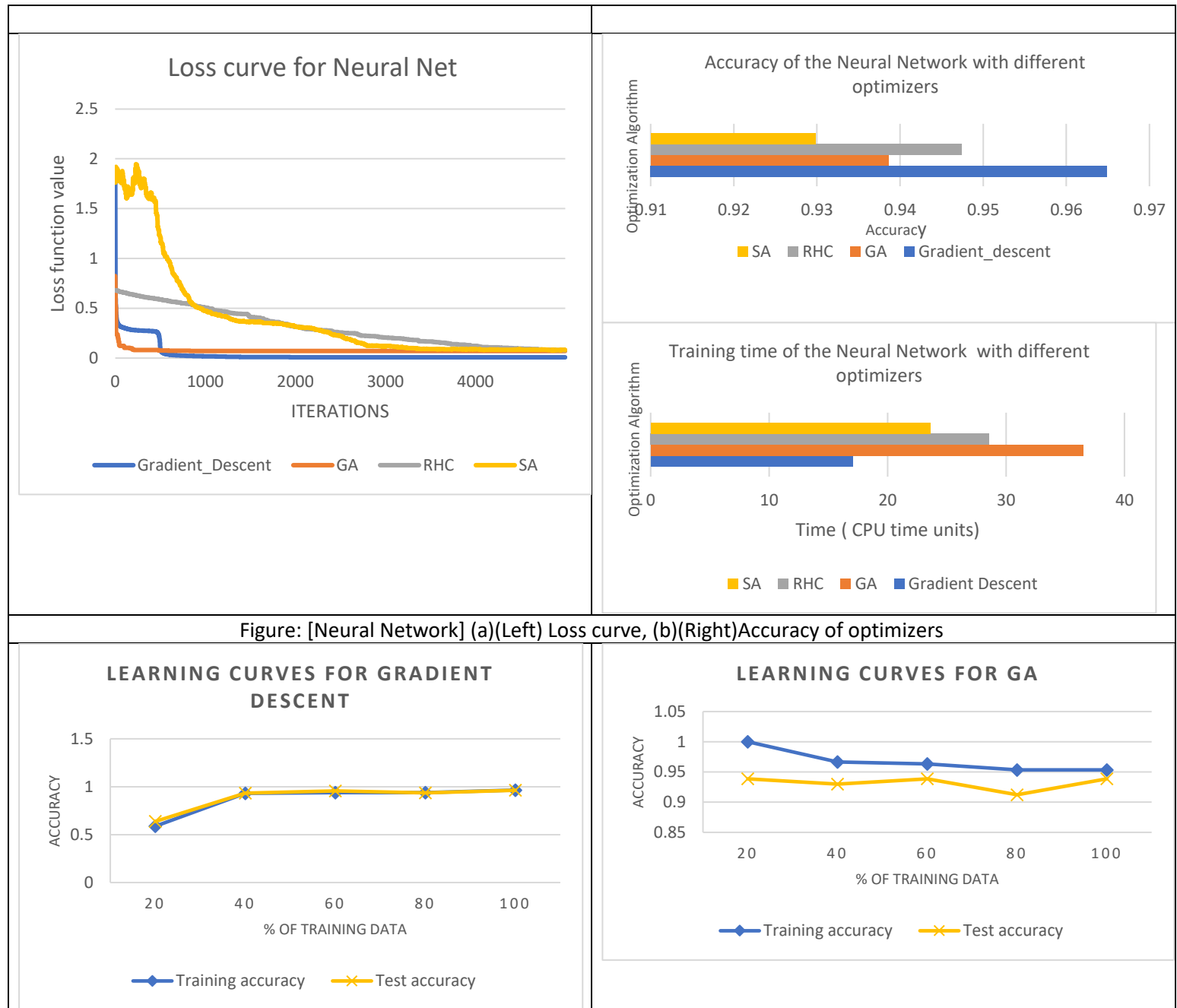
From the curves MIMIC comes up with a better score than any of the algorithms in much lesser iterations. In this problem the order in which the items are put in knapsack does not matter as opposed to TSP. Here MIMIC's understanding of the structure of the problem and its information propagation to the problem states closer to it enables the better performance. Recall, that this information sharing was difficult between closer states in TSP as the states depended upon the sequence of cities. Having said that this problem is easily converged by all the other 3 algorithms too but to an inferior optimum of 364. Expectedly MIMIC and GA takes large amount of time as compared to other 2 algorithms. The problem is much like a mathematical constraint and easy to optimize.

## Hyperparameter Analysis

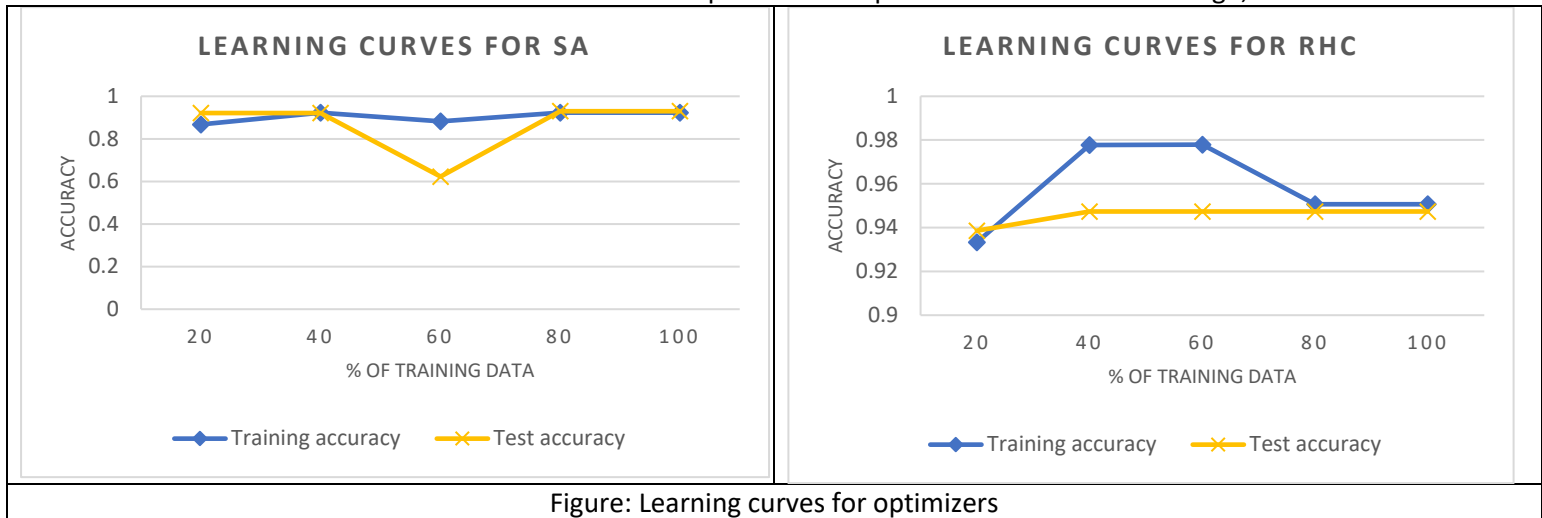
GA					
Population->	200	400		Any configuration of GA finds the optimum	
Mutation (below)					
0.1	364	364			
0.3	364	364			
0.5	364	364			
SA					
<p>Optimum value vs Initial temperature</p> 			Any initial temperature value finds the optimum, a fact which may not hold true with an increase problem size.		
MIMIC					
Population->	400	800	1600	3200	It is observed that for any configuration of MIMIC optimizer with population 1600,3200 it reaches the global optimum. Thus, the obvious choice was to go for the simplest and least time-consuming config with population 1600 and keep_pct 0.1. This observation also highlights that MMIC is best suited for this problem based on ease of convergence.
keep_pct(below)					
0.1	364	364	390	390	
0.5	364	364	390	390	
0.75	364	364	390	390	
RHC: restarts 40					

## 3.PART 2

In this part the breast cancer dataset from assignment 1 is predicted by neural network architecture using gradient descent, SA, RHC and GA as optimizers.







## Hyperparameter Analysis

For hyperparameter tuning there are many variables for the current neural net. I cast a wide net of hyperparameter using randomized search CV and then picked up parameters and ran a grid search CV to narrow it down to the most optimum set. Following are the parameters:

### Gradient Descent

**backprop\_params**={'hidden\_nodes': [(3,),(5, 5),(10, 10, 10),(5, 5, 5), (10, 10, 10), (20,20,20),(5)], 'activation': ['tanh', 'relu','sigmoid'], 'max\_iters': [x for x in range(0,iteration+1,1000)], 'learning\_rate':[0.001,0.005,0.01,0.1,0.5]}

**Best hyperparameter:** {'max\_iters': 5000, 'learning\_rate': 0.001, 'hidden\_nodes': (5, 5), 'activation': 'relu'}

It is observed that the NN is sensitive to learning rate. A higher learning rate fails to capture the optima. A sample of the parameter tuning is as follows:

param_max_it_param_learningparam_hidparam_actparams	split0_tes	split1_tes	split2_tes	split3_tes	split4_tes	split5_tes	split6_tes	split7_tes	split8_tes	split9_tes	mean_tes	std_tes	t_rank	test_score
5000 0.005 (5, 5) tanh {'max_iters': 5000, 'learning_rate': 0.005, 'hidden_nodes': (5, 5), 'activation': 'tanh'}	0.929825	0.877193	0.877193	0.964912	0.947368	0.929825	0.947368	0.964912	0.842105	0.928571	0.920937	0.039399	4	
5000 0.1 (3,) tanh {'max_iters': 5000, 'learning_rate': 0.1, 'hidden_nodes': (3, ), 'activation': 'tanh'}	0.894737	0.877193	0.929825	0.842105	0.947368	0.894737	0.894737	0.842105	0.842105	0.875	0.883991	0.03443	13	
5000 0.5 (5, 5) tanh {'max_iters': 5000, 'learning_rate': 0.5, 'hidden_nodes': (5, 5), 'activation': 'tanh'}	0.263158	0.473684	0.350877	0.368421	0.508772	0.385965	0.368421	0.280702	0.315789	0.285714	0.36015	0.076874	37	
5000 0.001 (5, 5) relu {'max_iters': 5000, 'learning_rate': 0.001, 'hidden_nodes': (5, 5), 'activation': 'relu'}	0.929825	0.982456	0.929825	0.964912	0.929825	0.947368	0.947368	0.982456	0.947368	0.964286	0.952569	0.019259	1	
1000 0.01 (10, 10, 10) relu {'max_iters': 1000, 'learning_rate': 0.01, 'hidden_nodes': (10, 10, 10), 'activation': 'relu'}	0.614035	0.614035	0.631579	0.631579	0.631579	0.631579	0.631579	0.631579	0.631579	0.625	0.627412	0.006966	28	
3000 0.1 (5,) tanh {'max_iters': 3000, 'learning_rate': 0.1, 'hidden_nodes': (5, ), 'activation': 'tanh'}	0.789474	0.754386	0.701754	0.77193	0.736842	0.719298	0.666667	0.824561	0.684211	0.839286	0.748841	0.055038	25	
1000 0.005 (5, 5) sigmoid {'max_iters': 1000, 'learning_rate': 0.005, 'hidden_nodes': (5, 5), 'activation': 'sigmoid'}	0.929825	0.859649	0.929825	0.894737	0.982456	0.912281	0.877193	0.929825	0.842105	0.892857	0.905075	0.038652	7	
4000 0.5 (5, 5) relu {'max_iters': 4000, 'learning_rate': 0.5, 'hidden_nodes': (5, 5), 'activation': 'relu'}	0.614035	0.614035	0.631579	0.631579	0.631579	0.631579	0.631579	0.631579	0.631579	0.625	0.627412	0.006966	28	
5000 0.005 (5, 5, 5) tanh {'max_iters': 5000, 'learning_rate': 0.005, 'hidden_nodes': (5, 5, 5), 'activation': 'tanh'}	0.912281	0.947368	0.859649	0.947368	0.947368	0.877193	0.912281	0.912281	0.842105	0.892857	0.905075	0.035323	6	

As shown in the figure the gradient descent optimizer is evaluated with different hyperparameter with 10-fold cross validation and the different configuration of the neural nets are ranked according to their performance.

### GA

**ga\_params**={'hidden\_nodes': [(3,),(5, 5),(10, 10, 10),(5, 5, 5), (10, 10, 10)], 'max\_iters': [x for x in range(0,iteration+1,1000)],pop\_size':[25,50], 'mutation\_prob':[.1,.3,.5], 'learning\_rate':[0.001,0.01,0.1]}

**Best hyperparameter:** {'pop\_size': 25, 'mutation\_prob': 0.1, 'max\_iters': 1000, 'learning\_rate': 0.001, 'hidden\_nodes': (3,)}

### RHC

**Best hyperparameter:** {'restarts': 2, 'max\_iters': 5000, 'learning\_rate': 0.1, 'hidden\_nodes': (5, 5), 'activation': 'sigmoid'}

### SA

**SA\_params**={'hidden\_nodes': [(3,),(5, 5),(10, 10, 10),(5, 5, 5), (10, 10, 10)], 'init\_temp':[1,10,50], 'learning\_rate':[0.001, .01,0.1,0.5], 'activation': ['tanh', 'relu','sigmoid']}

**Best hyperparameter:** :{'schedule': GeomDecay(init\_temp=1, decay=0.99, min\_temp=0.001), 'max\_iters': 4000, 'learning\_rate': 0.1, 'hidden\_nodes': (5, 5), 'activation': 'relu'}

## Performance

The dataset has 30 attributes; thus, it can be safely assumed that the optimal hyperplane has several local optima. Now we know that for such optima the optimizer should have a strong exploration capability. GA, gradient descent, RHC and SA rely upon crossover, stochasticity, random restarts and temperature schedule for this purpose. Out of these RHC is the weakest in exploration as it is a greedy approach which does not take a step if it results in a worse value of fitness function. When compared to RHC, the SA temperature allows it to accept inferior fitness function value at high temperature. Expectedly the slow convergence (in terms of iterations) of these 2 optimizers are shown in loss curve figure. RHC and SA takes around 9000 and 5000 iterations to converge. GA and gradient descent converge much earlier at around 500 iterations. In terms of wall clock time and accuracy gradient descent is the quickest and most accurate much ahead than all the other optimizers. GA works on the principle of starting with multiple set of parameters in form of population. Members of this population mutate and cross over to other neighbors thus exploring entirely new areas of parameter space, thereby resulting in a quick convergence (in terms of number of steps) in a complex problem. However, this process is totally oblivious to the fitness value. GA works on the 'survival of fittest' principle. At each time step the fitter members are kept and others discarded. However, all this process does take more wall clock time. The same behavior of GA is observed in the present analysis too as seen in loss curve and wall clock statistics. The clear winner in the present analysis is the gradient descent which apparently can strike an optimum balance between exploration and exploitation by a combination of its stochasticity and greediness towards minimizing loss value.

I manipulated my algorithm to generate the learning curves for the 4 optimizers as shown in figure.

For gradient descent: at 20% of training data the optimizer underfitted, however, as more training data was provided both testing and training accuracy increased and almost became equal and curve parallel indicating minimum bias and variance and sufficient data.

For GA: Similar as above but the testing curve seems to converge with training curve even at the full 100% training data indicating the model needs more training data. This is a reason of lower accuracy of GA on test set as seen in the figure.

For SA and RHC: the curves converge and run horizontally indicating minimum bias and variance and sufficient training data.

## BIBLIOGRAPHY

Code location: <https://drive.google.com/drive/folders/1UNfc-xwPxtpt8wjY1ZZKXZyifT7oc4gi?usp=sharing>

[1] <https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/>

[2] <https://mlrose.readthedocs.io/en/stable/source/tutorial3.html>