Randomized Optimization Assignment

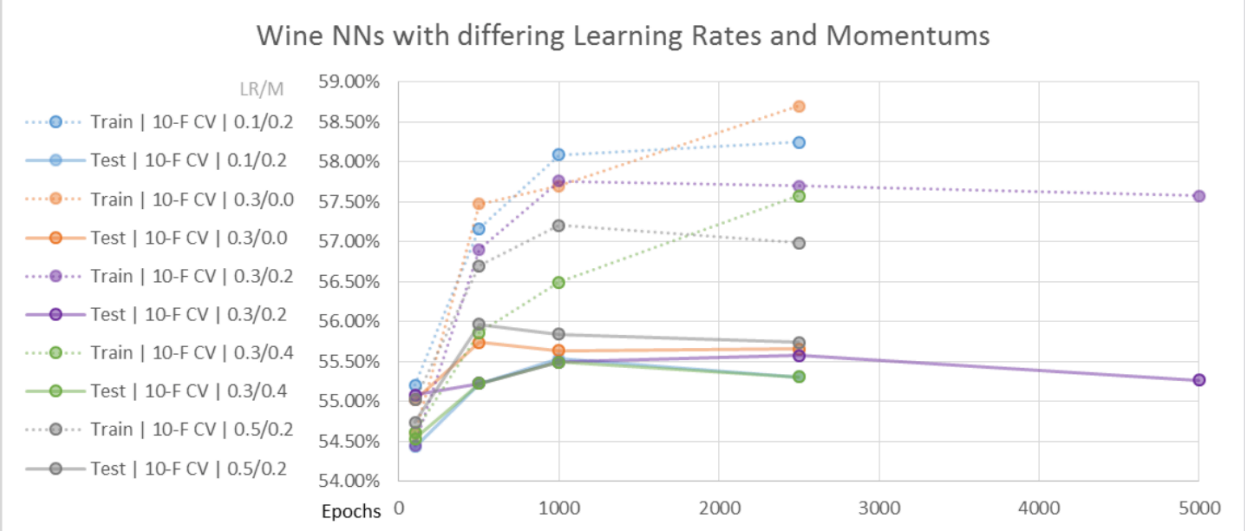
Yuanheng Wang
902905501

Dataset Description:

Dataset used for assignment 2 is from UCI Portuguese "Vinho Verde" wine dataset, particularly white wine data in this case. The task is to predict the quality of the wine, represented as an integer between 0 and 10. There are 4,898 instances and 12 attributes, including class. Each attribute is a numeric value including PH, density, alcohol, etc., whereas the quality is a subjective, nominal value: quality of the wine. The output has a normal distribution.

Assignment 1 recap:

The reason I want to use this dataset is because I want to explore the power of Neural Network in terms of multivariate output problem. Both the problems I used for assignment 1 is binary output. Therefore, I need to rerun the neural network for the part of homework 1 to come up with the best model. Below is just a short description and summarization of the process.



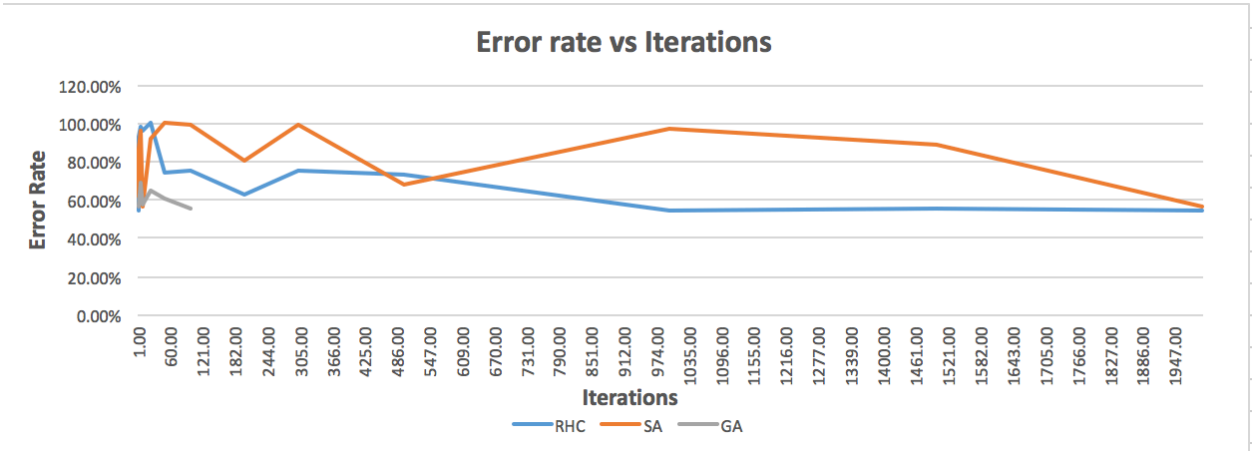
The learning rates and momentums are tuned to test performance of different combinations. Afterwards, by fixing Learning rate and momentum and limit Epochs to be 500 in order not to over-fit, by setting different number of hidden layers, 10 – 15 layers seem to be a reasonable range before trusting the data too much, which results in over-fitting.

Find good NN weights with optimization algorithms:

For this assignment, the input layer is 11 and output layer is 7, corresponding to 11 attributes and 7 possible outcomes (3-9) for the dataset. 10-fold cross validation is implemented to decide the best parameters for each algorithm. A 70/30 split is implemented to see the training error and test error of the corresponding model. The table below displays the model performance for different pairs of parameters. One thing to mention is that Randomized Hill Climbing doesn't have parameters so it's not showed in the table.

		Training Error	Testing Error	Training Time
Backpropogation	10 hidden layers	53.35%		
	200 hidden layers	46.53%	51.09%	38.539s
Simulated Annealing	Temperature:1E12 cooling down: 0.9	55.51%	57.483&	158.823s
	Temperature:1E12 cooling down: 0.95	59.22%	60.27%	173.337s
	Temperature:1E15 cooling down: 0.9	66.69%	67.35%	169.601s
	Temperature:1E15 cooling down: 0.95	56.68%	59.59%	164.847s
Genetic Algorithms	Population Ratio: 0.2 Mate Ratio: 0.02 Mute Ratio: 0.02	56.13%	57.69%	279.611s
	Population Ratio: 0.2 Mate Ratio: 0.02 Mute Ratio: 0.04	62.98%	65.71%	360.012s
	Population Ratio: 0.25 Mate Ratio: 0.02 Mute Ratio: 0.02	54.81%	56.05%	294.084s
	Population Ratio: 0.25 Mate Ratio: 0.02 Mute Ratio: 0.04	61.67%	62.86%	392.583s

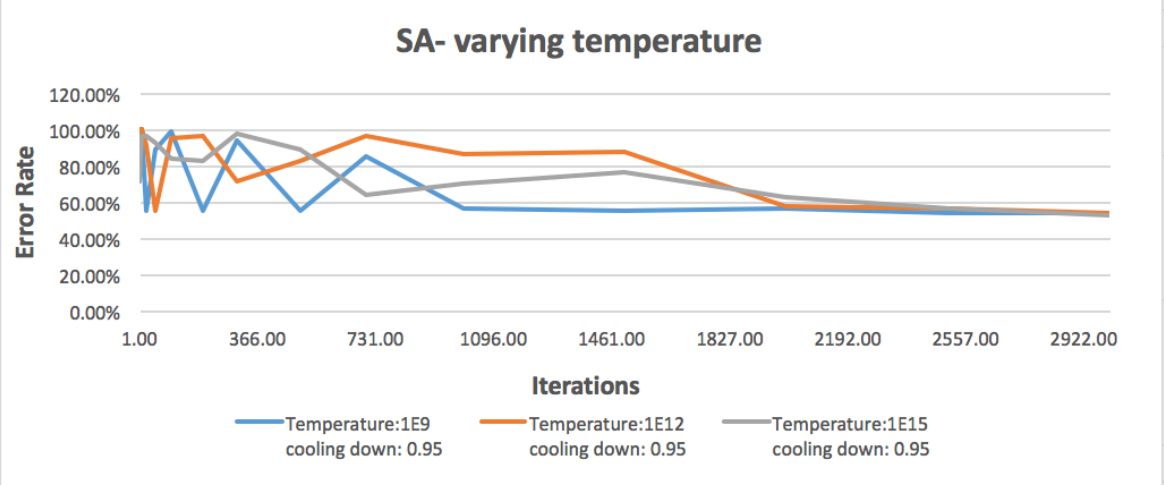
After optimal parameters are determined for each algorithm, I ran different numbers of iterations for each algorithm to see how efficient do each optimization method converge. Note for Genetic Algorithm, time of running increases exponentially with incrementing iteration numbers. For 100 iterations it takes over 20 minutes. More iterations are not applied to GA.

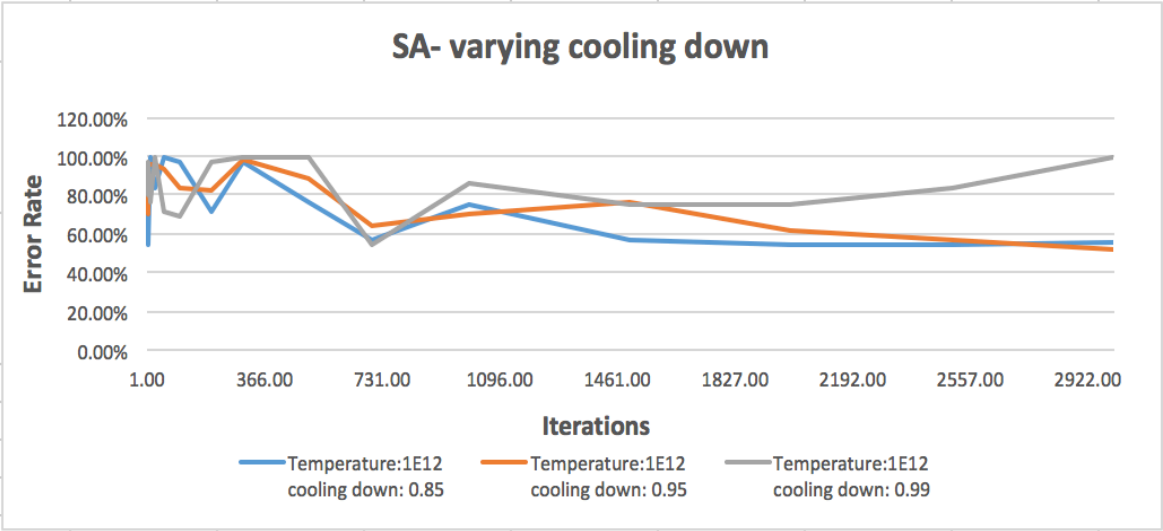


From the graph above, we see that genetic algorithm is very fast in converging. With single digit of iterations, it can reach around 55% of error rate. RHC converges slower, but manifests a steady decrease of error after several hundreds of iterations. To get to similar model accuracy, RHC takes about only one-tenth of the time that GA took. The long training time for GA is a result of the overhead of running the evaluation function as well as choosing instances to be mated, mutated, and so on. Furthermore, the search space for this problem is quite high, with

each attribute having a multitude of continuous values that it could take on. GA's do not work efficiently as the size of the search space increases. There don't seem to be many local optima in the wine dataset, which would have been something GA's are decent at overcoming. On the other hand, the absence of many local optima makes task very suited for Randomized Hill Climbing because it wouldn't be stuck. One thing to decrease the wall clock time for GA is to lower the population ratio, mutate ratio and mate ratio. If the problem is smooth, lowering these values would make trivial different regarding accuracy but will improve speed.

One counterintuitive observation from the previous graph is that Simulated Annealing is so fluctuating even with large iteration numbers. One possible suspect I have was because of the high temperature (1E12) that I initially selected. So I repeated the iterations steps for Simulated Annealing with distinct set of parameters pairs. Below are the performance graphs. One graph is to compare different temperatures: 1E9, 1E12 and 1E15. The other one is to compare the cooling rate: 0.85, 0.9, 0.95.





As a matter of fact, in the long run, the behavior of Simulated Annealing is similar to that of Randomized Hill Climbing and the resulting accuracy is pretty close. But it is highly stochastic with few number of iterations as it tries to apply this random exploration mechanism to escape local optima. This is a tradeoff between not getting stuck and high variance in terms of predicting.

From the graph, we see that changing the temperature of the algorithm has minor effect on the final result. However, a higher temperature does seem to fluctuate a little more at the beginning of the iterations. Higher starting temperatures allow a higher chance of choosing a neighbor that is worse, so we should expect to see a wider range of errors early on in the tests with higher temperatures.

The value of the cooling down controls how quickly the temperature decreases. A lower cooling factor means that the algorithm will converge more quickly, but could potentially get stuck in local optima more easily. A higher cooling factor allows more wiggle room, but will take longer to converge. When C = 0.99, the error fluctuated considerably, ever past iteration 3000 before converging. The tests with lower values of C converged more quickly but the error rate is a tiny bit higher. So depending on how much iteration budget one has, we may consider a lower temperature and bigger cooling rate for safe answer or the opposite to bet on a risky but potentially better answer if many iterations are allowed.

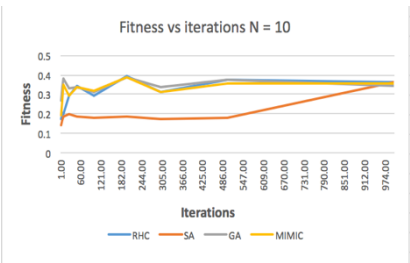
In summary, just as backpropogation, optimization algorithms can be used to generate neural network model with similar accuracy. For wine problem in particular, Randomized Hill Climbing is the best candidate to consider.

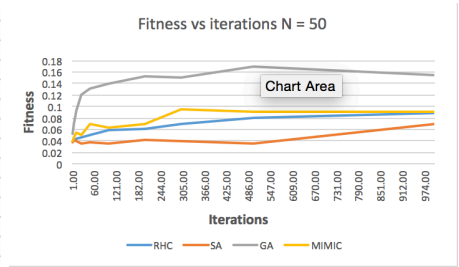
Traveling Salesman Problem:

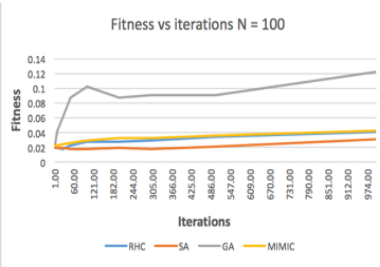
The traveling salesperson problem is an NP-hard problem. Its definition is as follows: A salesperson needs to visit *n* cities once and only once, and finishes the tour in the same place he started. The goal is to minimize the total travel distance of the salesman. As N increases, the complexity of this problem increases dramatically. For each N = 10, 50 and 100, I ran three trials and get the average result. Below is the relative data related with each run.

	Traveling Salesman							
	N = 10	Iterations = 3						
	RHC		SA		GA		MIMIC	
Trial	Fitness	Time	Fitness	Time	Fitness	Time	Fitness	Time
1	0.338218	0.052	0.339319	0.215	0.339319	0.149	0.318573	0.56
2	0.298413	0.049	0.30815	0.207	0.30815	0.15	0.302536	0.548
3	0.282527	0.049	0.274338	0.213	0.282527	0.144	0.2557278	0.562
Average	0.306386	0.05	0.307269	0.21166667	0.30999867	0.14766667	0.29227893	0.55666667
	N = 50	Iterations = 3						
	RHC		SA		GA		MIMIC	
Trial	Fitness	Time	Fitness	Time	Fitness	Time	Fitness	Time
1	0.113932	0.121	0.135473	0.278	0.147477	0.472	0.09165	16.977
2	0.134633	0.119	0.116581	0.263	0.153599	0.403	0.095156	18.308
3	0.123837	0.114	0.11258	0.262	0.164004	0.428	0.101612	16.793
Average	0.124134	0.118	0.12154467	0.26766667	0.15502667	0.43433333	0.09613933	17.3593333
	N = 100	Iterations = 3						
	RHC		SA		GA		MIMIC	
Trial	Fitness	Time	Fitness	Time	Fitness	Time	Fitness	Time
1	0.06885	0.192	0.074072	0.337	0.081676	0.868	0.038297	199.694
2	0.071146	0.185	0.065527	0.331	0.104198	0.809	0.043612	201.456
3	0.069668	0.189	0.0755449	0.332	0.096499	0.769	0.038601	214.474
Average	0.069888	0.18866667	0.07171463	0.33333333	0.09412433	0.81533333	0.04017	205.208
Parameters	iterations	20000	iterations	20000	iterations	1000	iterations	1000
			temperature	1.00E+12	population	200	samples	200
			cooling rate	0.95	mate rate	150	to keep	100
					mutate rate	20		

The code has already translated fitness function to the reciprocal of traveling distance, so the shorted the distance is, the higher fitness value we get. From the table we find Genetic Algorithm performs consistently well under all circumstances with reasonable amount of time. On the other hand, looking at graphs, we realize when the problem remains simple, SA, RHC and MIMIC all do pretty well, but as complexity increases, SA and RHC are more likely to be stuck in local optima while MIMIC does similarly poorly while consuming plenty of time because MIMIC estimates a lot more parameters and there is not really “structure” involved in TSP. The reason that GA converges fast and out performs other algorithms is likely to be that it contains information of previous solutions so it moves forward with slight changes on previous best generations. And unlike SA and RHC who only compare one value, it considers multiple solutions each time and keep improving. This is especially efficient when the search space becomes large.





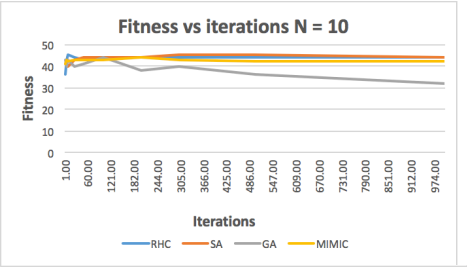


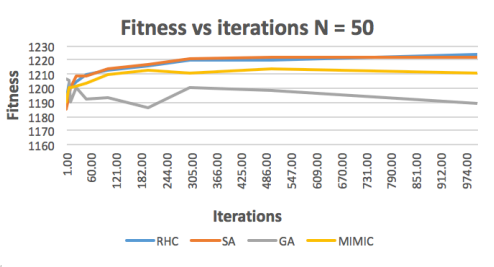
NQueens Problem:

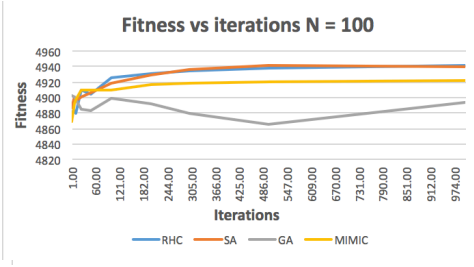
NQueens problem is defined as follows: on a chessboard of N by N size, you need to put N queens on certain positions on the board so none of them can attack each other. From the definition of Wikipedia, this problem has solutions for all natural numbers n except when n=2 or n=3. I set N to be equal to 10, 50, and 100, whose corresponding theoretical fitness value (for this specific program, fitness value is the pairs of queens that don't attack each other) should be C(10,2) = 45, C(50,2) = 1225, C(100,2) = 4950. Again, for each N, I run three trials and get the average result. Below is the data table.

	Nqueens Problem							
	N = 10	Iterations = 3		True Optimal = C(10,2) = 45				
	RHC		SA		GA		MIMIC	
Trial	Fitness	Time	Fitness	Time	Fitness	Time	Fitness	Time
1	44	0.006	44	0.002	39	0.039	43	0.028
2	44	0.007	44	0.003	39	0.016	43	0.025
3	44	0.007	45	0.007	38	0.017	41	0.029
Average	44	0.00666667	44.33333333	0.004	38.6666667	0.024	42.3333333	0.02733333
	N = 50	Iterations = 3		True Optimal = C(50,2) = 1225				
	RHC		SA		GA		MIMIC	
Trial	Fitness	Time	Fitness	Time	Fitness	Time	Fitness	Time
1	1217	0.029	1213	0.01	1192	0.06	1199	0.18
2	1216	0.03	1214	0.01	1198	0.067	1197	0.171
3	1215	0.03	1215	0.011	1184	0.059	1197	0.205
Average	1216	0.02966667	1214	0.010333333	1191.33333	0.062	1197.66667	0.18533333
	N = 100	Iterations = 3		True Optimal = C(100,2) = 4950				
	RHC		SA		GA		MIMIC	
Trial	Fitness	Time	Fitness	Time	Fitness	Time	Fitness	Time
1	4917	0.048	4921	0.006	4883	0.07	4893	1.059
2	4922	0.048	4924	0.006	0.074	0.074	4892	1.144
3	4919	0.048	4920	0.007	4878	0.071	4890	1.066
Average	4919.333333	0.048	4921.666667	0.006333333	3253.69133	0.07166667	4891.66667	1.08966667
Parameters	iterations	100	iterations	100	iterations	100	iterations	5
			temperature	1.00E+01	population	200	samples	200
			cooling rate	0.1	mate rate	0	to keep	10
					mutate rate	10		

As we observed from the data table, optimization algorithms don't always return the global optima. For this case, even simple complexity like N = 10 don't guarantee best solutions even with 1000 iterations. An interesting phenomena is that GA, while doing best for TSP problem, seems to be the most under-performed algorithm for NQueens problem. The reason being is that Genetic Algorithm takes the best performed cases and mutate them. As the positions of N queens are slightly changed, it is very likely the previous optima become worse. This is the reason why as the number of iterations increases, the fitness value decreases instead. There are two ways I think can give good values for this problem. One is to randomly search all the possible cases. RHC and SA is doing pretty well, both rendering high fitness value pretty fast. The reason is that they both randomly search one solution at a time without remembering the previous solutions. The other method is to actually consider the structure and position of each queen placed on the boards. One traditional way of solving NQueens is to use backtracking. For MIMIC, it actually needs to consider structure of the problem, which is kind of similar to backtracking because both needs to know the information of its parent, which is the position of the previous queen position to decide the possible candidate positions for current queen. Hence, MIMIC again is spending lots of time computing the parameters and structures. As iterations number grow bigger, MIMIC's performance should keep improving as the framework is understood.







Countones Problem:

The Count Ones problem is that of counting the number of "ones" in a bit string. Here fitness is measured by the number of points accurately able to be counted up to *n*, the correct number of ones. Once again, for each N = 10, 50 and 100, I ran three trials and get the average result. Below is the relative data related with each run.

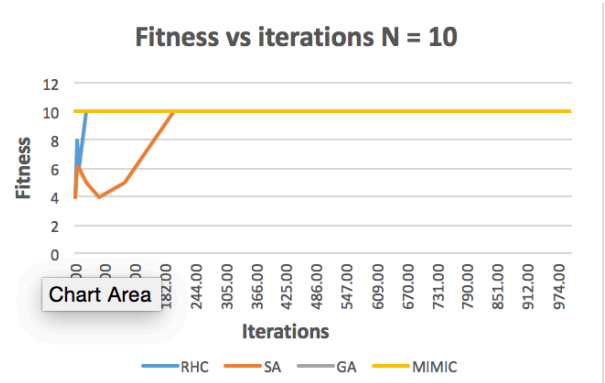
	N = 10	Iterations = 3		True Optimal = 10				
	RHC		SA		GA		MIMIC	
Trial	Fitness	Time	Fitness	Time	Fitness	Time	Fitness	Time
1	10	0.009	10	0.011	8	0.017	10	0.0324
2	10	0.01	10	0.011	9	0.017	10	0.319
3	10	0.01	10	0.011	8	0.017	10	0.33
Average	10	0.00966667	10	0.011	8.33333333	0.017	10	0.22713333
	N = 50	Iterations = 3		True Optimal = 50				
	RHC		SA		GA		MIMIC	
Trial	Fitness	Time	Fitness	Time	Fitness	Time	Fitness	Time
1	50	0.014	50	0.014	32	0.05	50	1.888
2	50	0.014	50	0.015	32	0.05	50	1.965
3	50	0.015	50	0.014	32	0.03	50	1.901
Average	50	0.01433333	50	0.01433333	32	0.04333333	50	1.918
	N = 100	Iterations = 3		True Optimal = 100				
	RHC		SA		GA		MIMIC	
Trial	Fitness	Time	Fitness	Time	Fitness	Time	Fitness	Time
1	100	0.022	100	0.019	59	0.052	100	6.807
2	100	0.023	100	0.02	57	0.05	100	6.471
3	100	0.22	100	0.022	54	0.054	100	6.651
Average	100	0.08833333	100	0.02033333	56.6666667	0.052	100	6.643
Parameters	iterations	20000	iterations	20000	iterations	1000	iterations	1000
			temperature	1.00E+12	population	200	samples	200
			cooling rate	0.95	mate rate	150	to keep	50
					mutate rate	20		

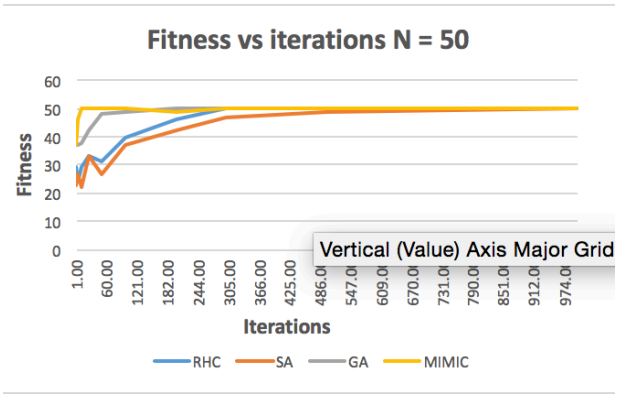
From the table, we find that all algorithms give very good values, although MIMIC take much longer to generate the results. It may seem like RHC and SA once again win. But when I dig into the fitness convergence efficiency, it is shown that MIMIC converges very fast, especially as the problem becomes more complicated. From the fitness vs iteration plots below, we can see that MIMIC converges to nearly optimal solution with less than 100 iterations. MIMIC is well suited to this problem because at each iteration it communicates far more than just a single point. By communicating information about the cost function and underlying

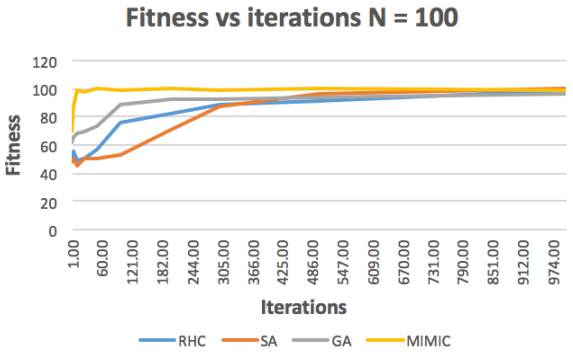
distribution from the previous iteration, MIMC doesn't waste time with poor solutions and will likely perform very well here. The structure of Countones problem is also comparatively simpler than that of NQueens problem, and this makes the calculation and exploration of structure for MIMIC much easier, therefore fewer iterations are needed.

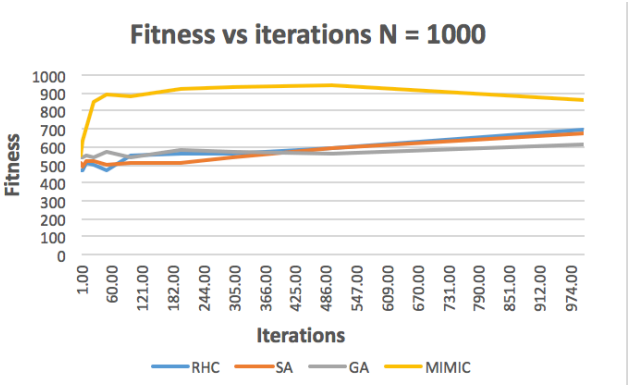
This brings an interesting point of when to use which algorithm. RHC and SA can get close similar performance as MIMIC with large number of iterations, although it maybe still be less time consuming as MIMIC has very high wall clock time. However, if the cost of each iteration is high for various reasons, it's smart to consider using MIMIC, especially when the problem gets more and more complex. (Refer to the Fitness vs Iterations N = 1000 graph).

Another notable thing is that when problems get complicated, unlike RHS and SA, GA doesn't grow as fast given more iterations. This is likely due to the property of mutation. When GA mutate the optimal solutions each time, it corrects some bits but it also makes some originally correct bits incorrect, which is a little bit similar to over-fitting as the algorithms believes in current optimal solutions too much.









Conclusion:

In general, randomized can be used to solve certain problems as well as being combined with supervised learning algorithms like neural network. However, there are many distinctions behind how each algorithm works and what types of problems are suited for which algorithm. RHS and SA are very efficient in terms of each iteration, but they only look at one solution at once. Also their slightly different local optima escape mechanism can also result in difference

for solving the problems. It's not obvious at the scope of this assignment but it definitely worth exploring. Genetic algorithms do consider multiple solutions at a time, but its mutation can be either a bless or a curse regarding the type of problems we try to solve. The mutation attribute can mess up, for example, the position of NQueens. Nonetheless, GA is still relatively faster than MIMIC algorithm. MIMIC does extremely well when structure of the problem matters. After certain numbers of iterations, MIMIC is like a radar that detects and understands the entire problem space, with the cost of intense computing and high wall clock time. Thus, we should be wise about choosing algorithms to tackle various challenges and trying to understand the problem may be a good first step.