CS 7641 Machine Learning — Assignment 2: Randomized Optimization

Robert Bobkoskie

I. INTRODUCTION

The goal of this project is to explore the use of Randomized Optimization algorithm (OA): Randomized Hill Climbing (RHC), Simulated Annealing (SA), a Genetic Algorithm (GA) and MIMIC to assign weights to a neural network. The assignment is broken into two parts:

- Part one will use OA instead of backpropagation to assign weights to classify the bankP marketing problem[1] from the first assignment. Since the bankP marketing problem proved to be trivial for OA, an additional problem[2] (abalone) was considered for this assignment.
- Part two will employ the OA methods to maximize a fitness function for the problems: FourPeaks, Knapsack, FlipFlop and CountOnes.

The OA methods were evaluated using ABAGAIL[3]. Graphing and dataset preprocessing was accomplished using python: sklearn, pandas, numpy and matplotlib.pyplot.

II. Problems

The first problem (bankP), **Bank Marketing Data Set**, was obtained from the **UCI Machine Learning Repository**. The data originates from a direct marketing campaign of a Portuguese banking institution. The marketing campaign was initiated to sell a bankP term deposit (product) to clients. After preprocessing, the problem consisted of: 3811 instances with 20 attributes of the client: [education, marital status, existing loans, day of week, etc.]. For evaluation, the problem was divided into test and training sets of 1905 instances for each. The objective is to classify (label), and predict the outcome, ‘client subscribed’: ('yes') or ('no').

The second problem (abalone) was the Abalone Data Set, was obtained from ABAGAIL. The data is a collection of abalone attributes, e.g., diameters and weights. After preprocessing, the problem consisted of: 4177 instances with 7 attributes. For evaluation, the problem was divided into test and training sets of 2088 instances for each. The objective is to classify (label), and predict the age of the abalone.

Although part one of the project called for using a problem from the first assignment (bankP), even the most challenging problem from that assignment proved too easy for the OA. To evaluate these methods more accurately, I chose to evaluate a second problem (abalone). abalone proved to be a better challenge, and provided more interesting results. For each problem, either the optimization hyper-parameters or the number of training iterations were adjusted, and error captured until the most accurate results for each OA was achieved. To provide graphical support, graphs of the number of errors vs. the number of training iterations were plotted for each run. Performance data was also collected for the test error and test time.

To test the fitness functions in part two, I chose problems that would highlight each of the OA methods: [FourPeaksTest: RHC, SA], [KnapsackTest: GA], [FlipFlopTest: SA], [CountOnesTest: MIMIC]. SA required two problems, as the FourPeaksTest problem was used to highlight the similar behavior of RHC and SA with the tuning of SA hyper-parameters. It should be noted that I chose problems to evaluates the fitness functions that required a minimal tuning of the optimization hyper-parameters, e.g., the problems were a good match for the OA. In addition, h-param tuning was minimized, as a baselined for the optimization hyper-parameters was obtained from the first part of this project. For each problem, the optimization hyper-parameters were held constant and the 'n value' for each problem was increased from ten to three hundred by increments of ten. Both the time to evaluate the problem as well as the fitness score were captured for each run. For each problem, the 'n value' determines the scope (size) of the problem, e.g., for the KnapsackTest, the 'n value' is the number of items in the pack. Evaluating each problem with different 'fitness values' gives a good estimate of the fitness achieved for each OA across a range of problem sizes.

To provide graphical support, graphs of the fitness score vs. the 'n value' were plotted for each run. To get a sense of performance, graphs of the time (logarithmic) vs. the 'n value' were also plotted for each run.

For both parts of this project, I modified ABAGAIL to output the data to *.csv files. I create the graphs by importing these csv files and plotting with matplotlib.pyplot. The csv files used for the graphs are included in the zip file submission under the 'OUTPUT DATA' folder.

III. Tune the Optimization Hyper-Parameters

The data was split into two equal sized sets for training and testing. H-params were evaluated on the training set and then tested to see how the model generalized the problem. The objective of tuning the optimization hyper-parameters and adjusting the training iterations was to achieve the highest accuracy on generalizing to the test set. RHC was used as a baseline, thus the optimization hyper-parameters were not tuned for this OA. Testing was not exhaustive, e.g., optimization hyper-parameters were chosen at boundary points and the models evaluated. For example, when evaluating SA, the initial temperature (t) and cooling (c) variables were tested in a range of: [`1E8 < t < 1E11, 0.05 < c < 5.0`]. For GA, the population size (p), mate (m) and mutate (x) were tested in a range of [`200< p < 2000, 100 < m < 1000, 10 < x < 100`]. The number of training iterations was also adjusted to balance performance with accuracy. Once an accuracy threshold was achieved, no further evaluation of the optimization hyper-parameters was conducted.

IV. DISCUSS THE RESULTS

For both problems, it became apparent that an upper bound on accuracy was achieved, and increasing the number of training iterations, or tuning the optimization hyper-parameters did not improve accuracy. However, this was not the case for training time. It's obvious that increasing the number of training iterations correlates to increased training time, but less obvious are the influence optimization hyper-parameters have on training time. This was most evident with GA, where increasing p, m, or x had a strong correlation to increased training time. It was observed that increasing p, m, and x by an order of magnitude, increased the training time by an order of magnitude. For SA, training time stayed relatively constant under varying optimization hyper-parameters. As expected, testing time stayed relatively constant for both GA and SA when optimization hyper-parameters were adjusted.

I was surprised and disappointed that bankP was so trivial when weights were assigned using OA (table 3). Recall that in assignment 1 (see table 1), when using a neural network with BP, in fact for all learners, the most accurate results achieved for bankP was ~90%. With minimal training iterations using OA, 100% accuracy was achieved. This necessitated a second problem for evaluation using OA. The abalone problem was run using BP and OA from ABAGAIL. One again, the results (table 2) for abalone were surprising and disappointing. It was disappointing that the maximum accuracy for abalone was ~90% for OA. Even with adjusting the optimization hyper-parameters to boundary (extreme) values and maxing out the number of training iterations (`num train iters == size of train set`), I was not able to achieve a higher accuracy on the test set. What surprised me was the performance of BP using ABAGAIL. Though the not statistically significant, the accuracy obtained with BP was greater than when using OA. The training time using BP also outperformed. This outperformance on accuracy with BP may be due to ABAGAILS accuracy calculation that's derived from the difference between the predicted and actual values.

Table 1: Summery of Accuracy for all learners from the first assignment: SVM, KNN and Neural Network. Only the data under BANK are relevant, as these show the results of bankP evaluated with BP in assignment 1.

Table 1 **Summary of Accuracy Assignment 1**	**Dec Tree**		**Dec Tree (Rnd Frst)**		**Dec Tree (Bag ged)**		**Dec Tree (Add Bst)**		**Dec Tree (Grd Bst)**		**KNN**		**SVM**		**Nrl Net**	
	IRIS	**BANK**	**IRIS**	**BANK**	**IRIS**	**BANK**	**IRIS**	**BANK**	**IRIS**	**BANK**	**IRIS**	**BANK**	**IRIS**	**BANK**	**IRIS**	**BANK**
Accuracy Train %	*96*	*90*	*93*	*89*	*95*	*90*	*96*	*89*	*88*	*85*	*96*	***90***	***97***	*90*	*97*	*90*
Accuracy Test %	*96*	*90*	*93*	*90*	*89*	*90*	*96*	*89*	*90*	*86*	*96*	***91***	***97***	*90*	*97*	*90*
*X_val (T / F / B) **	*T*	*F*	*T*	*T*	*T*	*F*	*T*	*T*	*T*	*F*	*B*	***T***	***T***	*T*	*T*	*T*

* x_val (T) means best score achieved with cross validation enabled, x_val (F) means best score achieved with cross validation disabled, x_val (B) means best score x_val (enabled == disabled).

Table 2: Summary of results from OA and BP on the abalone problem. Note that BP appears to produce the most accurate results. Also note the effect of increasing the population size, mate and mutate (hyper-parameters) has on the training time for GA.

Table 2	**100***	**100**	**1000**	**2088**	**2088**	**Hyper-Parameters and Notes**
Part 1	**5****	**50**	**5**	**5**	**50**	*GA (population size,mate, mutate)* *SA temp, cooling)*
ABALONE BP:						***Note that BP is the most accurate with good training time. More hidden layers appear to be slightly more accurate than more train iterations.***
Train Time (s)	***0.002***	***0.002***	***0.001***	***0.002***	***0.002***	
Test Time (s)	***0.054***	***0.040***	***0.021***	***0.035***	***0.059***	
# Misclassified	***170***	***166***	***174***	***163***	***172***	
# Correct	***1918***	***1922***	***1914***	***1925***	***1916***	
Accuracy Test %	***91.858***	***92.050***	***91.667***	***92.193***	***91.762***	
ABALONE RHC:						***Note more train iterations are slightly more accurate than more hidden layers.***
Train Time (s)	***1.895***	***6.088***	***13.13***	***40.913***	***126.63***	
Test Time (s)	***0.019***	***0.040***	***0.011***	***0.022***	***0.042***	
# Misclassified	***185***	***185***	***171***	***165***	***172***	
# Correct	***1903***	***1903***	***1917***	***1923***	***1916***	
Accuracy Test %	***91.140***	***91.140***	***91.810***	***92.098***	***91.762***	
ABALONE SA:						***SA: (1E11, .95), Note the poor performance (accuracy) when the number of train iterations is low.***
Train Time (s)	***1.768***	***5.886***	***12.92***	***38.107***		
Test Time (s)	***0.023***	***0.044***	***0.008***	***0.020***		
# Misclassified	***1903***	***1903***	***177***	***173***		
# Correct	***185***	***185***	***1911***	***1915***		
Accuracy Test %	***8.860***	***8.860***	***91.523***	***91.715***		
ABALONE SA:						***SA: (1E8, .05), Note the improved performance of SA when the initial temp and cooling vars are lowered from (1E11, .95). Also note that with lower temp, the performance is like RHC. This will be explored more in part 2 of this project.***
Train Time (s)	***1.505***	***6.192***				
Test Time (s)	***0.022***	***0.049***				
# Misclassified	***185***	***185***				
# Correct	***1903***	***1903***				
Accuracy Test %	***91.140***	***91.140***				
ABALONE SA:						***SA: (1E20, 5.0), Note accuracy not improved with higher init temp and cooling vars.***
Train Time (s)		***6.111***			***125.27***	
Test Time (s)		***0.044***			***0.040***	
# Misclassified		***185***			***185***	
# Correct		***1903***			***1903***	
Accuracy Test %		***91.140***			***91.140***	
ABALONE GA:						***GA: (200, 100, 10), Note the impact hyper-parameters have on increasing train time for GA.***
Train Time (s)	***32.206***	***249.68***	***146.36***	***709.63***		
Test Time (s)	***0.016***	***0.041***	***0.010***	***0.020***		
# Misclassified	***185***	***185***	***185***	***185***		
# Correct	***1903***	***1903***	***1903***	***1903***		
Accuracy Test %	***91.140***	***91.140***	***91.140***	***91.140***		
ABALONE GA:						***GA: (2000, 1000, 100), Note the longer train time for GA as the population size, mate and mutate hyper-parameters are increased. The train time increased ~one order of magnitude over GA: (200, 100, 10).***
Train Time (s)		***2322.66***				
Test Time (s)		***0.036***				
# Misclassified		***185***				
# Correct		***1903***				
Accuracy Test %		***91.140***				
ABALONE GA:						***GA: (20, 10, 5), Note improvement in train time for GA as the population size, mate and mutate hyper-parameters are decreased. The improvement in train time does not negatively impact accuracy.***
Train Time (s)		***32.189***			***691.2***	
Test Time (s)		***0.039***			***0.048***	
# Misclassified		***185***			***185***	
# Correct		***1903***			***1903***	
Accuracy Test %		***91.140***			***91.140***	

* Number of training iterations, ** Number of hidden layers.

Table 3: Summary of results from OA and BP on the bank problem. Note that BP appears to produce the most accurate results. Also note the effect of increasing the population size, mate and mutate (hyper-parameters) has on the training time for GA. Since accuracy was trivial, not much experimentation with the h-param, or the number of training iterations was required. One interesting observation is that lower values for the SA temperature and cooling variables produced the most accurate results. I would have expected with higher temperature values, SA be more likely to accept and take downward steps, like a random walk, and thus produce more accurate results. With low temp, SA behaves like RHC, at least in terms of accuracy.

Table 3 **Part 1**	**100*** **5****	**100** **50**	**Hyper-Parameters and Notes** *GA (population size,mate, mutate)* *SA temp, cooling)*
BANK BP:			***Note that BP is the most accurate with good training time. More hidden layers appear to be slightly more accurate than more train iterations.***
Train Time (s)	***0.002***	***0.002***	
Test Time (s)	***0.011***	***0.094***	
# Misclassified	***0***	***0***	
# Correct	***1905***	***1905***	
Accuracy Test %	***100.00***	***100.00***	
BANK RHC:			***Note accuracy, and that with low init temp, SA behaves like RHC, at least in terms of accuracy.***
Train Time (s)	***1.696***	***14.775***	
Test Time (s)	***0.024***	***0.093***	
# Misclassified	***0***	***0***	
# Correct	***1905***	***1905***	
Accuracy Test %	***100***	***100.00***	
BANK SA:			***SA: (1E11, .95), Note the poor performance (accuracy) when the init temp and cooling are too large for the problem.***
Train Time (s)	***2.644***	***16.294***	
Test Time (s)	***0.023***	***0.103***	
# Misclassified	***1905***	***1905***	
# Correct	***0***	***0***	
Accuracy Test %	***0***	***0***	
BANK SA:			***SA: (1E8, .5), Note SA accuracy improved over (1E11, .95).***
Train Time (s)	***2.856***	***15.242***	
Test Time (s)	***0.025***	***0.089***	
# Misclassified	***73***	***0***	
# Correct	***1832***	***1905***	
Accuracy Test %	***96.168***	***100***	
BANK SA:			***SA: (1E8, .05), Note with cooling value lowered from 0.5 to 0.05, accuracy of 100% is achieved.***
Train Time (s)	***2.962***		
Test Time (s)	***0.023***		
# Misclassified	***0***		
# Correct	***1905***		
Accuracy Test %	***100***		
BANK GA:			***GA: (200, 100, 10)***
Train Time (s)	***83.972***	***759.682***	
Test Time (s)	***0.023***	***0.103***	
# Misclassified	***0***	***0***	
# Correct	***1905***	***1905***	
Accuracy Test %	***100***	***100***	
BANK GA:			***GA: (20, 10, 5), Note improvement in train time for GA as the population size, mate and mutate hyper-parameters are decreased. The improvement in train time does not negatively impact accuracy.***
Train Time (s)		***91.384***	
Test Time (s)		***0.101***	
# Misclassified		***0***	
# Correct		***1905***	
Accuracy Test %		***100***	

* Number of training iterations, ** Number of hidden layers.

Figure 1 (below): Number of error vs number of training iterations for 100 training iterations on the abalone problem. Note that in experiments where the training error converged to a low value (BP, RHC, GA), the accuracy on the test results was good, ~90%. Note that SA did not converge to a low training error, thus the accuracy on the test results was low.

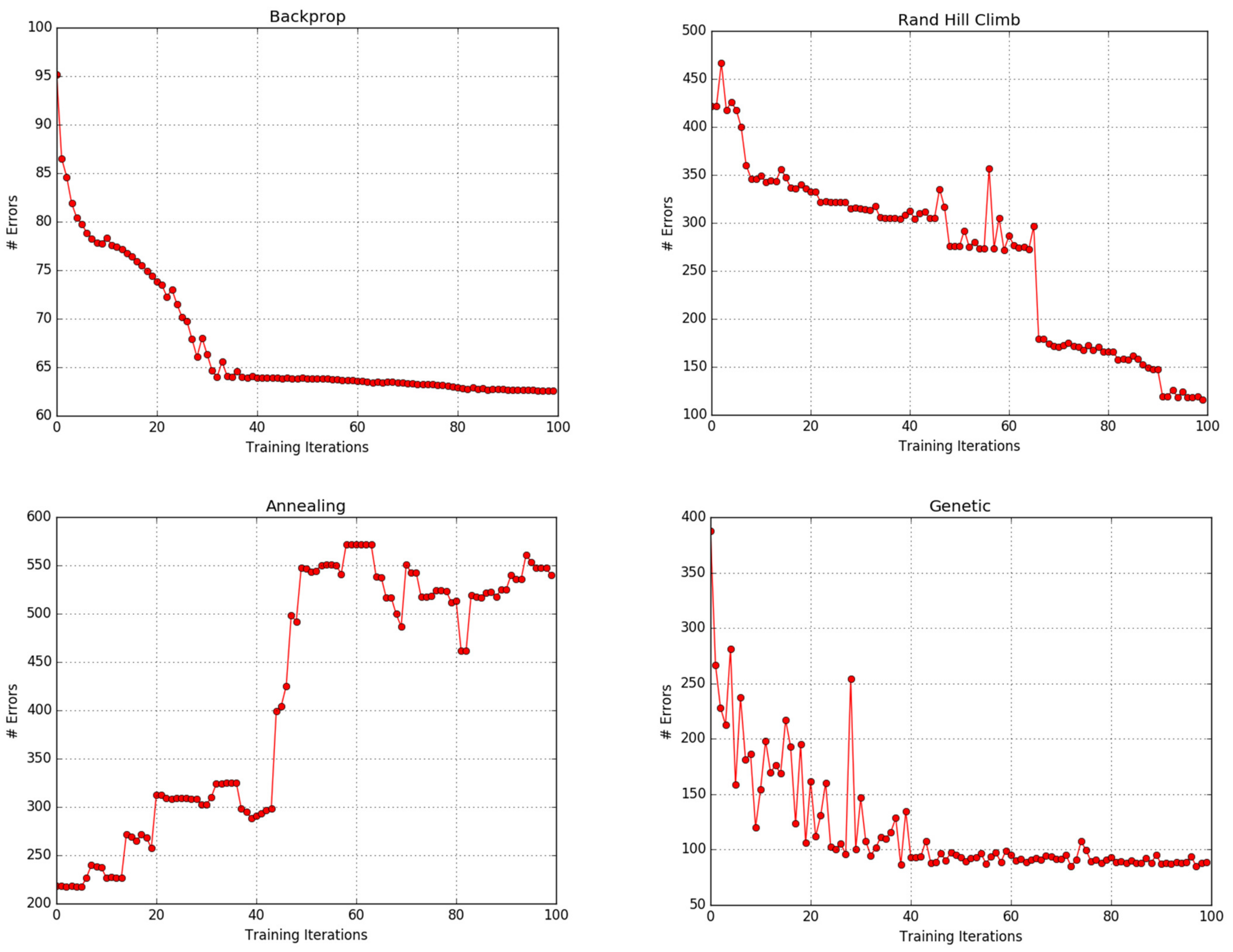


Figure 1a (right): Lowering the init temp and cooling vars for SA improved accuracy (converged to a low training error). Also note the similarity of this graph to that of RHC in figure 1.

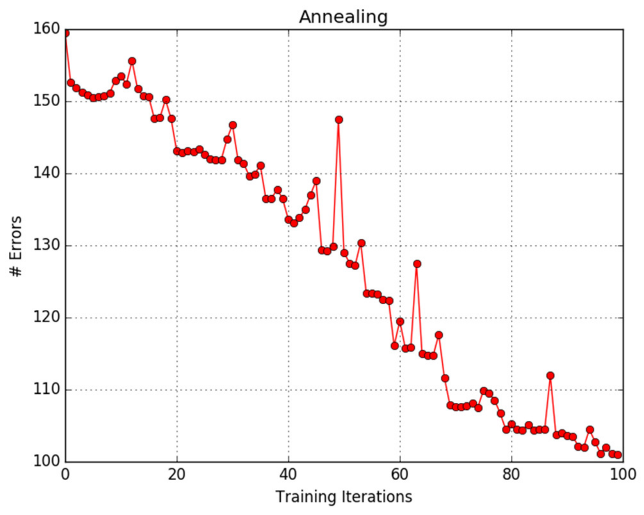


Figure 2 (below): Number of error vs number of training iterations for 100 training iterations on the bank problem. Note the graphs are like the abalone problem, however, the training error BP, RHC and GA converges faster too a lower value for the error. Again, due to the higher values for temp and cooling, SA behaves as a random walk, and converges to a high training error.

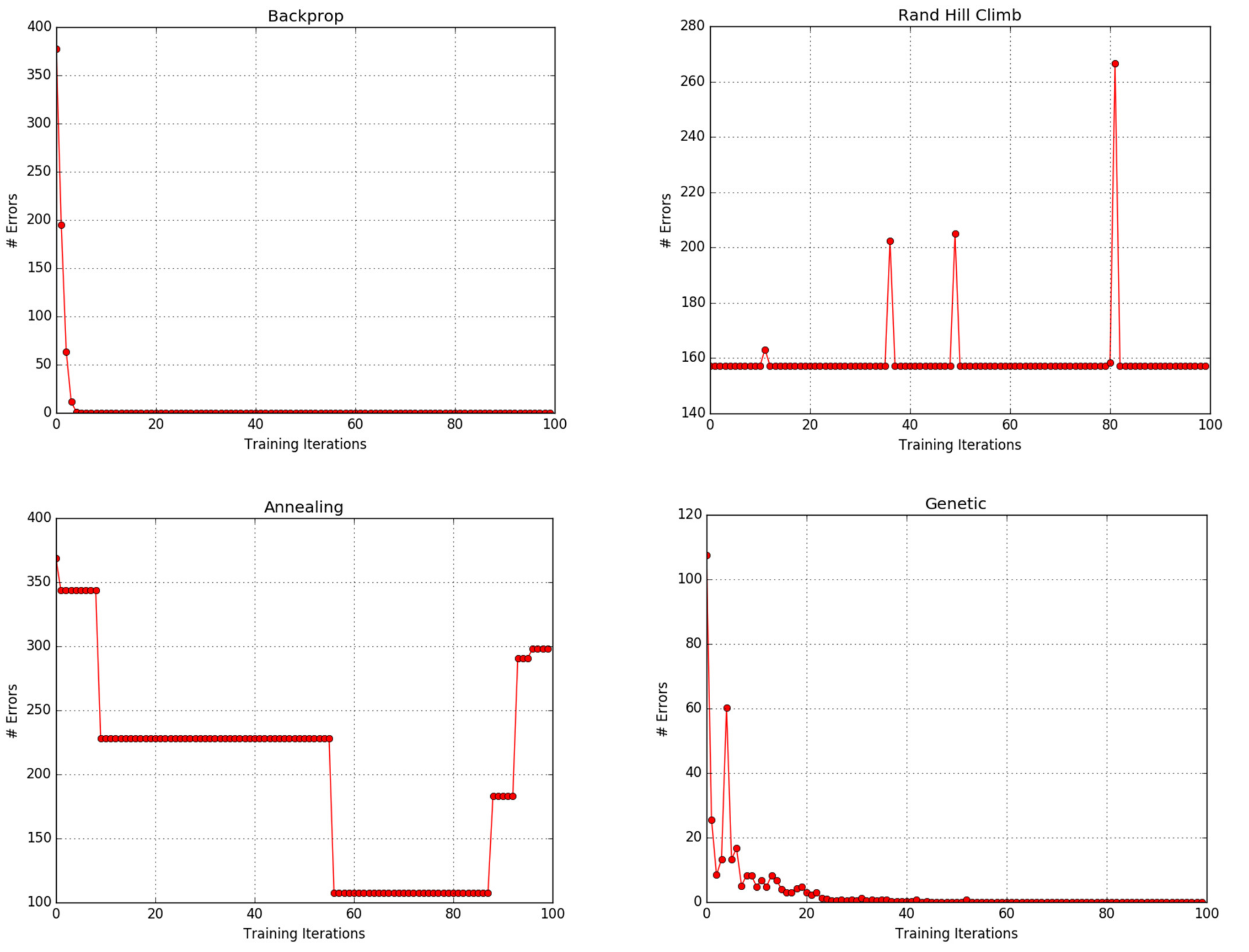
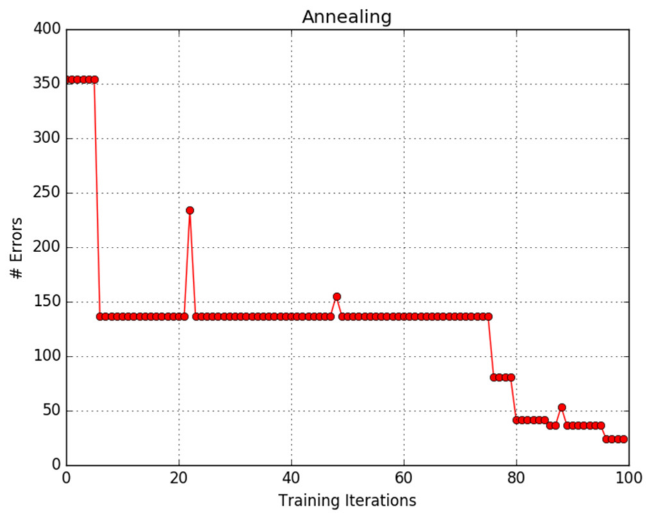


Figure 2a (right): Lowering the init temp and cooling vars for SA improved accuracy (converged to a low training error).



For part 2, highlighting the interplay of OA methods with the optimization problems: [FourPeaksTest: RHC, SA], [KnapsackTest: GA], [FlipFlopTest: SA], [CountOnesTest: MIMIC] is best presented graphically. For each problem, two graphs are shown: fitness score vs. fitness values, and test time (logarithmic) vs. fitness values.

Figure 3: FourPeaksTest, RHC and SA returned the highest fitness score over a range [10, 300] of 'n values'. RHC took the most time to train.

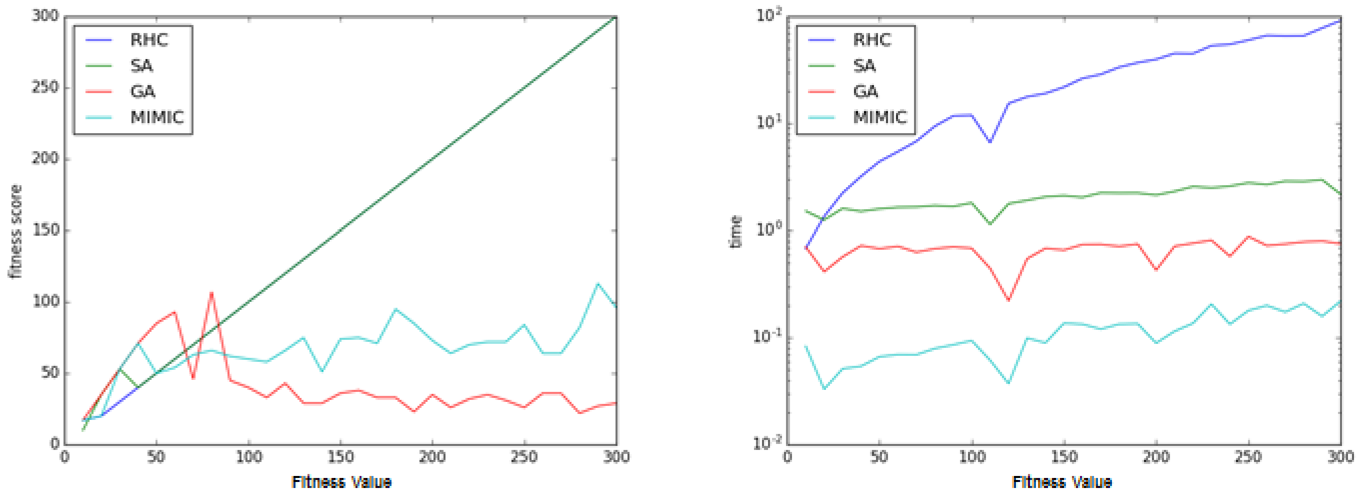


Figure 4: KnapsackTest, GA returned the highest fitness score over a range [10, 300] of 'n values'. RHC took the most time to train.

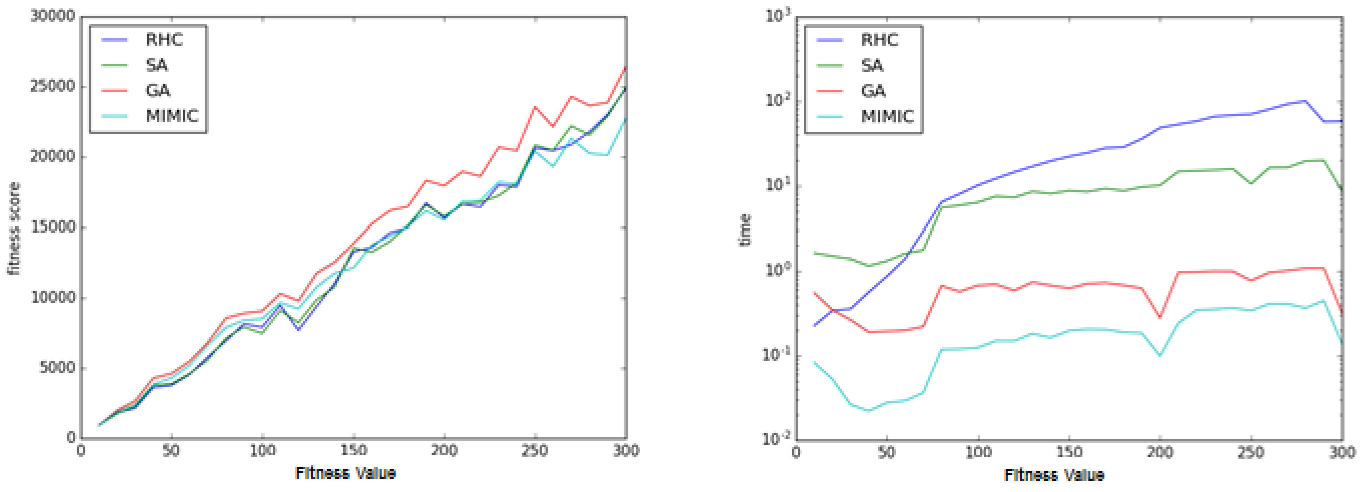


Figure 5: FlipFlopTest, SA returned the highest fitness score over a range [10, 300] of 'n values'. RHC took the most time to train.

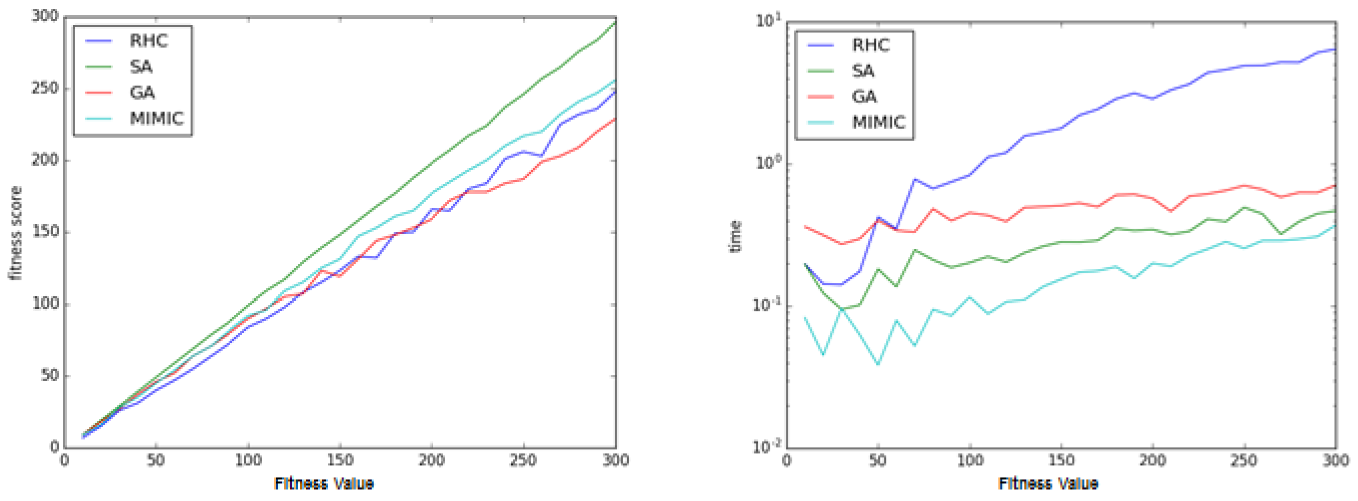
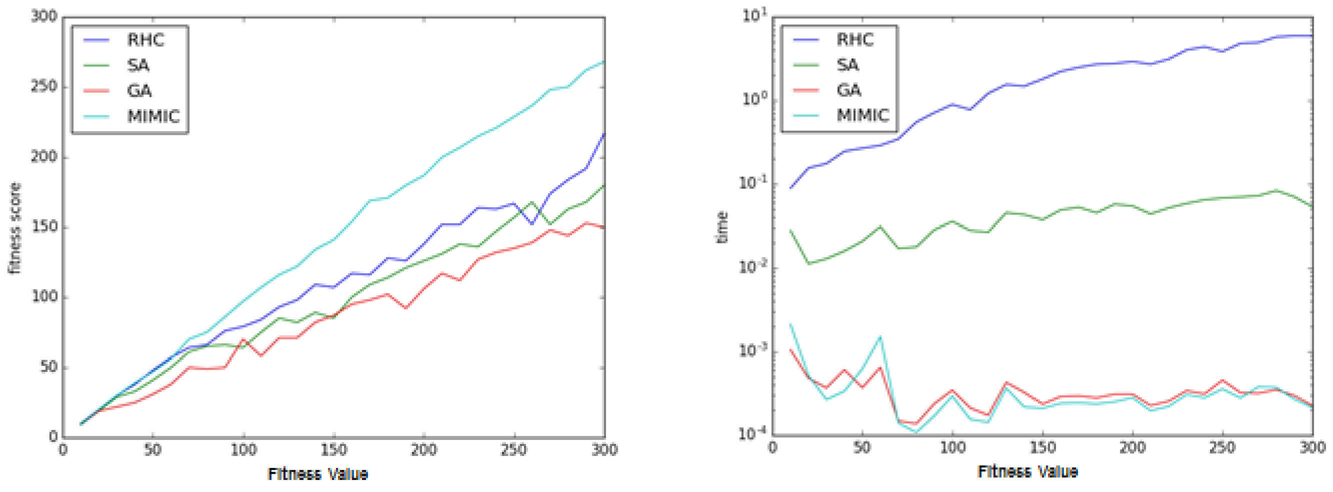


Figure 6: CountOnesTest, MIMIC returned the highest fitness score over a range [10, 300] of 'n values'. RHC took the most time to train.



V. CONCLUSION

Surprises and disappointments were noted in the discussion of results. One interesting observation from part 1 is the relationship of SA to RHC. When the initial temperature and cooling values are set low enough, SA behaves like RHC. This was shown both in the results, and graphically. I was surprised that higher values for temp and cooling would not produce greater accuracy for SA. Overall, for the brute force method employed by RHC worked surprisingly well in optimization. This was observed for both the Neural Network (weight optimization), as well as the maximizing the fitness function for the optimization problems. One caveat with RHC, it was not always the most efficient in terms of run-time. Further analysis would likely show RHC takes far more iterations to converge than the other optimization algorithms.

---

[1] The UCI repository: https://archive.ics.uci.edu/ml/datasets/Bank+Marketing
[2] The UCI repository: https://archive.ics.uci.edu/ml/datasets/Abalone
[3] ABAGAIL: http://abagail.readthedocs.io/en/latest/