Assignment 2: CS7641 - Machine Learning

Saad Khan

October 24, 2015

1 Introduction

The purpose of this assignment is to explore randomized optimization algorithms. In the first part of this assignment I applied 3 different optimization problems to evaluate strengths of optimization algorithms. In addition to this, in the second part of this assignment I applied the optimization algorithms to find optimal weights for a feed forward Neural Network for Pima Diabetes data sets used in assignment 1 (description is in assignment 1 report).

2 Optimization Algorithms

Randomized Optimization algorithms are used for obtaining the global maximum for difficult problems that cannot be derived. This report will go over 4 of these optimization algorithms namely Randomized Hill Climbing (RHC), Genetic Algorithm (GA), Simulated Annealing (SA) and MIMIC.

2.1 Randomized Hill-Climbing (RHC)

Randomized Hill climbing is a searching technique that looks for the global optimum by moving towards the next higher elevation neighbor until it reaches the peak. With random restarts (as the name implies), this algorithm randomizes its original starting position to eliminate getting stuck at a local optimum and increases the probability to choose a restarting position that will bring it closer to the global optimum.

Randomized Hill-Climbing, for optimization problems and neural network training was run as part of the java code that was implemented using ABAGAIL and its performance is compared to other algorithms in the later sections.

2.2 Genetic Algorithm (GA)

Genetic Algorithms provide solutions to the optimization problems using techniques inspired by natural evolution such as mutation, crossover, etc. These algorithms initially start from a population of candidate solutions and continuously evolve to a better solution over no. of iterations by eliminating non adequate sections of the population and retaining the parts of the population that exhibit the best traits. The best traits can be determined by mutating and mating (crossover) different parts of the population. One of the drawbacks of genetic algorithms is that they do not scale well with complexity, i.e. where the number of elements which are exposed to mutation are large there is often an exponential increase in search space size.

In this report genetic algorithm is also implemented using ABAGAIL as part of the java code and is evaluated against other algorithms in sections to come.

2.3 Simulated Annealing (SA)

Simulated Annealing is inspired by metallurgy where metals are heated to high temperature and then slowly cooled to increase ductility. The algorithm considers a neighboring state s' of the current state s, and probabilistically decides the step towards state s' or staying in state s. These probabilities ultimately lead the system to move to states of lower energy. This iterative process is repeated until the system reaches a state that is the optima or a given computation budget is exhausted. By starting from a state that is initially a worse solution than its current state the algorithm increases the probability to reach the global optima slowly but surely.

We will see later how simulated annealing fares against other optimization algorithms. Again simulated annealing implementation in this assignment was done using ABAGAIL and run in Eclipse for this report.

2.4 MIMIC

Mutual-Information-Maximizing Input Clustering (MIMIC) algorithm finds optima by estimating probability densities. Most of the optimization algorithms lack structure and MIMIC uses knowledge of this structure as a guide for randomized search through the solution space. It attempts to communicate information about the cost function obtained from one iteration of the search to later iterations of the search directly.

MIMIC implementation used in this assignment is again done using ABAGAIL. I will attempt to take advantage of the structure MIMIC maintains in order to compare its performance against other optimization algorithms in the sections to follow.

3 Optimization Problems

This section covers the three optimization problems that were considered to evaluate the strength of different optimization algorithms. ABAGAIL along with Java was used to run the algorithms in Eclipse. With some modification in the code I also logged computation times for all four algorithms. Iteration recorded for each optimization problem were from 1 up till 5000 with varying intervals (raw data is in the accompanied MS excel document). 3 separate runs were conducted, for each iteration value, for all these algorithms and average of the results were used for the fitness function values. The optimization problems are as follows:

3.1 Continuous Peaks Problem

3.1.1 Introduction

The continuous peaks problem is a continuation of the 4 peaks and the 6 peaks problem and is a good study in figuring out the highest peak (global maxima) vs the subsidiary peaks (local maxima). A classical issue with most optimization approaches, such as hill climbing, is that we can easily find local maxima but not the global ones and this problem exploits that well by furnishing continuous local peaks separated by considerably large plateaus. In this problem we will try to compare Randomized Hill Climbing, Genetic Algorithm, Simulated Annealing and MIMIC. The final goal in this problem is to reach the red 'x' and avoid getting stuck at any blue 'x' as shown in the Figure 1 below.

3.1.2 Analysis

Continuous peaks problem was implemented in java using ABAGAIL and all 4 algorithms were tested for increasing number of iterations. Fitness function values along with computation times

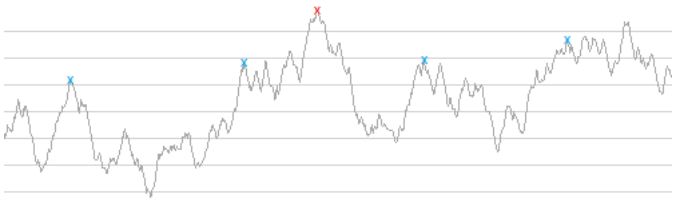

Figure 1: [Continuous Peaks] - Goal is to get to red 'x'

were noted and are shown in Figure 2 below.

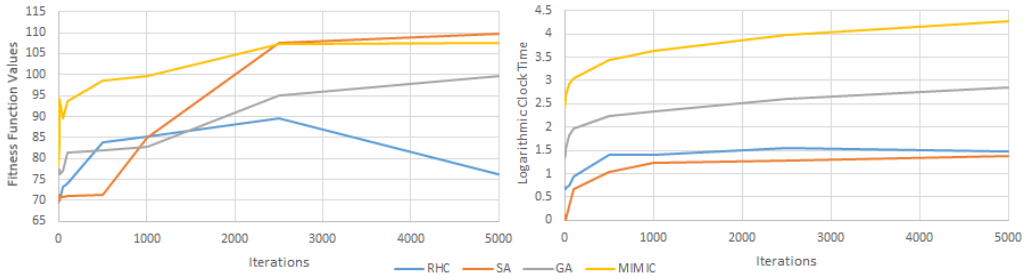

Figure 2: [Continuous Peaks Problem] - LEFT: Function, RIGHT: Computation Time

Using this problem I wanted to highlight the performance of simulated annealing over other algorithms but initially for smaller no. of iterations MIMIC was outperforming simulated annealing. It was above 2500 iterations, where simulated annealing started to kick in but only got slightly better results than MIMIC. Genetic algorithm did not do so well while randomized hill climbing got battered badly for higher iterations which was expected.

Logarithmic computation times are plotted in Figure 2 [RIGHT]. As iterations for MIMIC increase, the time was exponentially increasing so I decided to compare time in logarithmic scale. It is clearly evident that simulated annealing is the quickest of them all.

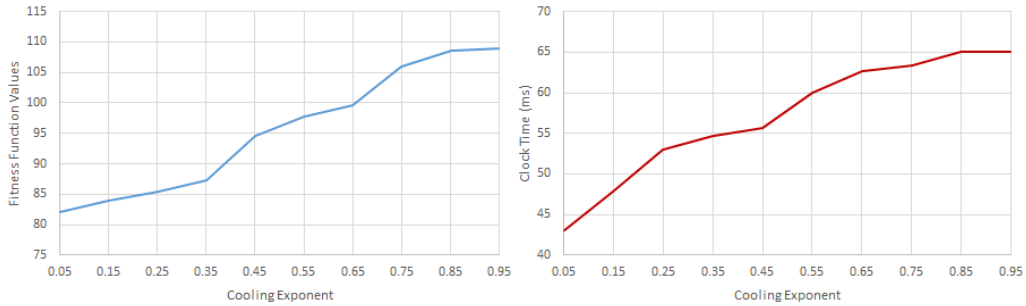

Figure 3: [Continuous Peaks Problem] - LEFT: Function, RIGHT: Computation Time

Further tweaking was done on the simulated annealing algorithm to see which configuration worked best for the problem at 5000 iterations (this is where the best performance occurred in Figure 2). Using cooling exponent parameter, function values were plotted shown in Figure 3 [LEFT] along with computation time in Figure 3 [RIGHT]. Although, the algorithm took extra time to compute for higher cooling exponent (CE) values but overall performance improves with increasing CE. Best configuration for SA was with CE: 0.95 at 5000 iterations.

3.2 Knapsack Problem

3.2.1 Introduction

The knapsack problem is an interesting problem in combinatorial optimization: For a no. of elements, each with a mass and a value, we determine the no. of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. Backpackers face this problem when given a fixed-size knapsack and have to fill it with the most valuable items for the hike. Here we intend to determine the most optimized strategy to fill up the sack.

The java version of this problem is implemented using ABAGAIL. The maximum possible volume of the sack was set at 3200. Maximum weight and volume of 50 units per item was used with 40 unique items having 4 copies each. Slight modification to the code was done to log computation times and the experiment was run in Eclipse. Plot in Figure 4 [LEFT] shows how the algorithms fared against each other.

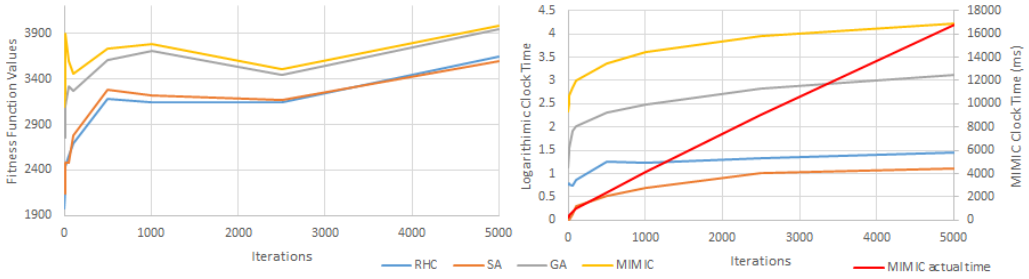

Figure 4: [Knapsack Problem] - LEFT: Function, RIGHT: Computation Time

3.2.2 Analysis

I chose this problem to accentuate the capabilities of MIMIC. Figure 4 [LEFT] shows, although, just slightly better than genetic algorithm, MIMIC performs best of all the algorithms for all no. of plotted iterations. Simulated annealing and randomized hill climbing perform rather poorly and fail to converge.

Figure 4 [RIGHT] shows the logarithmic computation time comparison (again for ease of representation I used logarithmic scale). Secondary axis in this plot shows actual average time taken for MIMIC iterations (red line). The time taken for computation linearly increases with increasing no. of iterations.

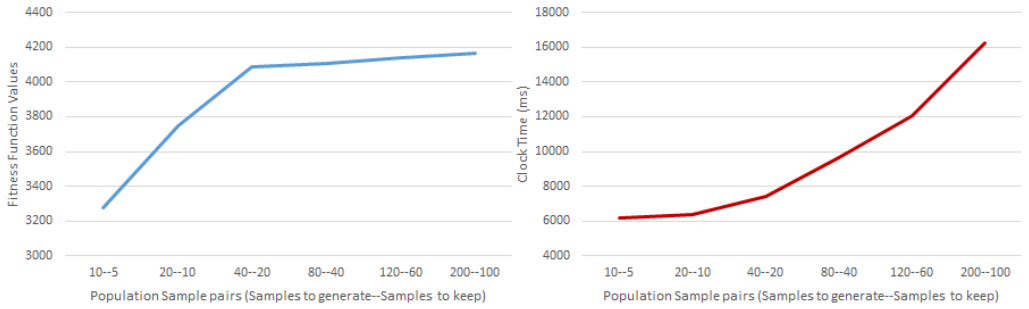

Figure 5: [Knapsack Problem] - LEFT: Function, RIGHT: Computation Time

Additional performance evaluation of the MIMIC algorithm was performed by changing the population size and sample size. This is shown in Figure 5 where it can be seen that function values are highest for population size of 200 with exponential increase in computation time. In this

particular problem, if we do not have any constraint of time we should definitely go with population/sample size combination: 200/100, however, with time constraints the population/sample size of 40/20 would suffice as it is not that far off in function value and takes lesser time to compute.

3.3 Traveling Salesman Problem

3.3.1 Introduction

Traveling salesman problem is a classical example of an NP-hard problem where the goal is to find the shortest round-trip between 'n' cities/points while visiting each node just once. Traveling salesman problem also occurs in every day life. Planning optimal routes between cities/points is a crucial task for trucking companies or a hiker who wants to go for an expedition. In this problem we do not care about the direction in which we are traveling, we can take any path but make sure we visit each node only once.

To illustrate the problem in this assignment, I have implemented Traveling salesman code using ABAGAIL and the means of measurement in this problem is using reciprocal of distance (d), i.e. 1/d. The algorithm producing larger 1/d on average has smaller distance calculation, hence providing a better solution to the problem. Below is the algorithmic comparison plot in Figure 6 [LEFT].

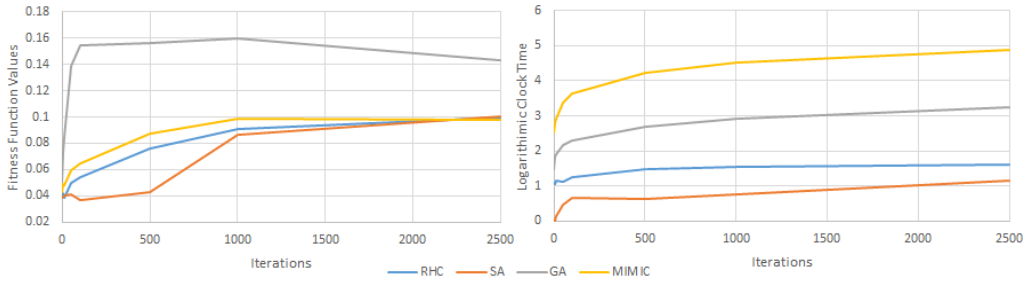

Figure 6: [Traveling Salesman Problem] - LEFT: Distance Function, RIGHT: Computation Time

3.3.2 Analysis

This problem was chosen to highlight abilities of genetic algorithm to find optimal solution with mutations over several generations. Here it is by far the best performing algorithm on the problem with all the other algorithms failing to find an optimal solution even over larger iterations.

Computation time is again shown in log scale in Figure 6 [RIGHT] which shows that genetic algorithm takes more time than Simulated annealing and randomized hill climbing but less than MIMIC. The best performance for genetic algorithm comes at the 1000th iteration.

Further investigation pertaining to the performance of the genetic algorithm was done by changing the following parameters while keeping the no. of iteration constant:

- population size: Population which equates to survival of the fittest;
- toMate: is crossover which represents mating between individuals;
- toMutate: is mutation which introduces random modifications.

Different combinations of the above parameters were used in order to identify the best configuration that applies to this problem. Figure 7 below shows the trend.

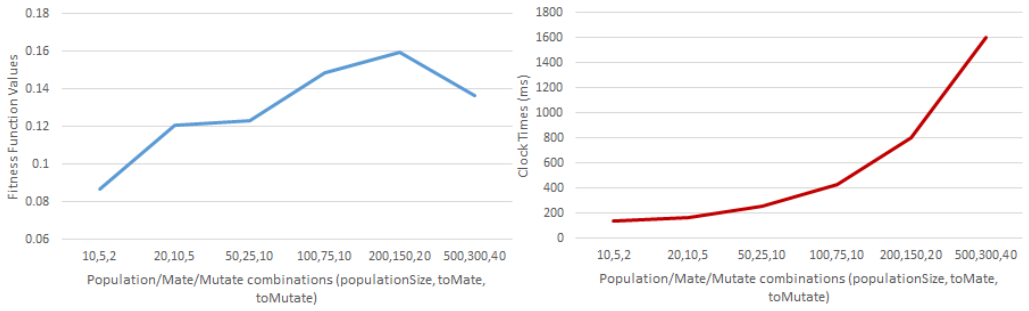

Figure 7: [Traveling Salesman Problem] - LEFT: Function, RIGHT: Computation Time

The plot shows clearly that best performing configuration is with population size at 200 toMate at 150 and toMutate set at 20. This combination also takes lesser time to compute as it seems that computation time is some what proportional to the toMate parameter.

4 Neural Network with Optimization Algorithms

4.1 Introduction

This section of the assignment focuses on finding how 3 optimization algorithms [Randomized Hill Climbing, Simulated Annealing and Genetic Algorithm] perform to find the optimal weights of the neural network for one of the data sets used in assignment 1 in this case the Pima Diabetes data set.

4.2 Data set

I have used the Pima Diabetes data set for the purpose of this part of the assignment. The goal is to classify a patient as either type 2 diabetic or not diagnosed with type 2 diabetes. This data set has 768 instances with patients being classified as having type 2 diabetes to class '1' and not having type 2 diabetes to class '0'.

4.3 Implementation

4.3.1 Code for determining number of iterations

Java was used to implement the code to compare all 3 algorithms. The code was executed in Eclipse using ABAGAIL. First the AbaloneTest.java code in the ABAGAIL package was used for the diabetes data set to generate sum of squared error plot to identify the maximum no. of iterations required for testing the performance of the algorithms.

4.3.2 Code of logging accuracy and computation time

After that a modified version of the code posted by a class mate on piazza [discussion no: 324] was used to compare the performance of the algorithms for different iterations. Three runs for each configuration and iteration were executed and mean of the results was used for the analysis. For this purpose I split the data set 70/30 into training and testing set using Weka Explorer. This piece of java code along with test set accuracy also provides training set accuracy which will be used in analysis later on.

4.3.3 Layer configuration used for the neural network

Another point to be noted here is that I used Weka for my first assignment and the optimal hidden layer combination for this data set was 7 input layer nodes, 3 hidden layer nodes and 1 output layer node for each classification. So all the experiments performed in this section are based on a 7-3-1(1 for each class) network.

4.3.4 Re-running neural network in Weka with percentage split for comparison

For assignment 1 I used cross-validation for neural network training but for comparison with the optimization algorithms I re-ran neural network in Weka CLI with 70/30 percentage split logging training and test set accuracy along with computation time.

4.3.5 Algorithm evaluation methodology

The evaluation of individual algorithms was done against the test set and the best performing combination for each of these algorithms in terms of classificaiton accuracy and computation times for up to 5000 iterations were noted.

Then I compared best performing combination of all 3 algorithm with result I obtained using back-propagation on neural networks (in assignment 1) in terms of correctly classified labels and computation times.

5 Algorithmic Analysis

5.1 Number of Iterations

Figure 8 shows below that sum of squared errors for all 3 algorithms when run on the diabetes data set. To avoid the variations in the error curves due to randomization/jittery effect I used running mean for 5 previous iterations and created a more presentable plot.

It can clearly be seen that for all 3 algorithms, curves start to flatten out to some extent after the 1000th iteration and keeps on the same trend till the 5000th iteration. So I decided to test all the algorithms individually up till 5000 iterations in order to find the best possible combination for each of these algorithms which gives the highest classification accuracy. This is done using the modified java code as mentioned above (details are in the README file).

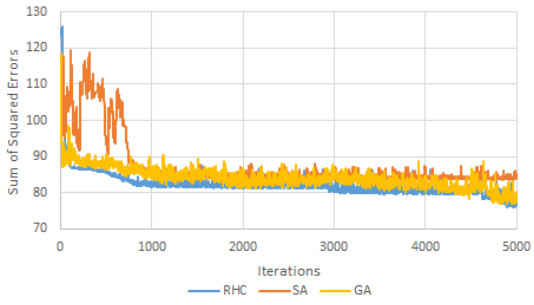

Figure 8: [Running average of Sum of Squared Errors]

5.2 Randomized Hill Climbing

As discussed in the optimization problem section, Randomized hill climbing (RHC) works on determining which direction to go to find the global optima by randomly restarting at new

places. This should work well in case of neural networks as it works on a similar principal of gradient descent to find the optima. Figure 9 below shows how randomized hill climbing performs against the test set.

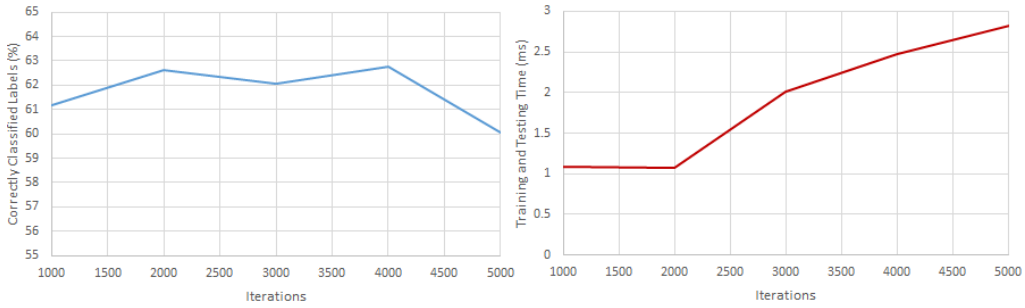

Figure 9: [Randomized Hill climbing] - LEFT: Correct Labels, RIGHT: Computation Time

It can be seen in the plot that correctly classified labels on the test set at 4000 iterations are 62.8%. This is very slightly better than 62.6% at 2000 iterations. Although, 4000 iterations take 1.5 ms more to compute in terms of training and testing times combined, I think best performance of the algorithm comes with 4000 iterations. We will compare this performance with other algorithms in the later sections.

5.3 Simulated Annealing

Simulated Annealing (SA) was tested against the diabetes data set with different combinations of the cooling exponent (CE) in the same way as I did in the first part of this assignment.

Figure 10 [LEFT] below shows how the algorithms fares against the testing set with different values of cooling exponents (CE) (starting from 0.15 with steps of 0.2 up till 0.95) in terms of correctly classified labels. Along with this, Figure 10 [RIGHT] shows the combined training and testing times for different iterations and cooling exponents.

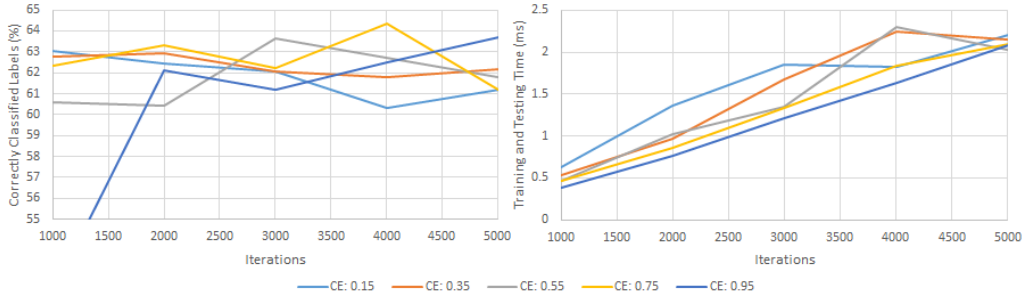

Figure 10: [Simulated Annealing] - LEFT: Correct Labels, RIGHT: Computation Time

Best performance in correctly classifying labels for simulated annealing comes at 4000 iterations with cooling exponent of 0.75. This is shown by the yellow curve in the plot. The computation time curve for this value of cooling exponent, again in yellow, is linearly increasing and is only slower to the configuration with cooling exponent of 0.95 at 4000 iterations. Now let us look at Genetic Algorithm and see how it fares against the test set.

5.4 Genetic Algorithm

The performance of genetic algorithm was evaluated by trying different combinations of population size/toMate/toMutate parameters as done in previous section of this assignment.

Figure 11 below shows how genetic algorithm classifies labels for increasing iterations with different combinations of population size, toMate and toMutate parameters.

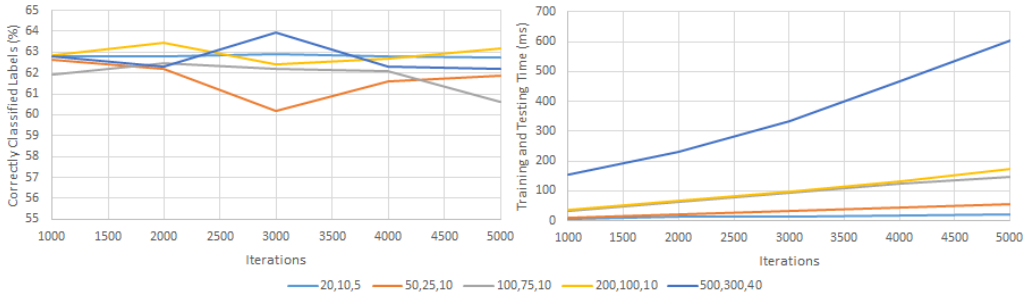

Figure 11: [Genetic Algorithm] - LEFT: Correct Labels, RIGHT: Computation Time

The highest percentage of correctly classified labels in this case came with combination of 500, 300, 40 at 3000 iterations. The 2nd best performance of the algorithm came with 200, 100, 10 combination at 2000 iterations.

Analyzing both the classification percentages and computation times, combination of 500, 300, 40 is a better choice for the network.

5.5 Comparative Analysis

This section compares the best performing configurations for each of the algorithms against the diabetes test set. Randomized hill climbing did not have any options or parameters to choose from so I kept the plot as it is. For simulated annealing I used the test set classifications curve with cooling exponent of 0.75. The combination that I selected for genetic algorithm is population size/toMate/toMutate : 500, 300, 40. Figure 12 below shows how the three algorithms compare with each other.

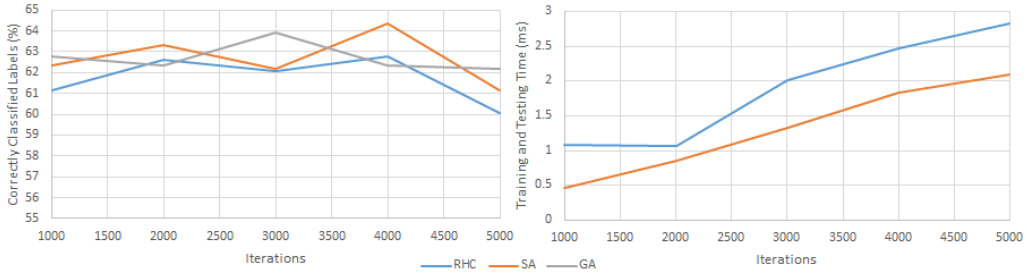

Figure 12: [Algorithmic Comparison] - LEFT: Correct Labels, RIGHT: Computation Time (RHC and SA only)

It can be seen in Figure 12 [LEFT] that simulated annealing at 4000 iterations is the best algorithm at correctly classifying patients as being positive with type 2 diabetes or not. Genetic Algorithm at 3000 iteration is a close 2nd with randomized hill climbing performing poorly.

For computation times in Figure 12 [RIGHT], I have excluded time for genetic algorithm as it was orders of magnitude higher than the computation time for simulated annealing and randomized hill climbing, i.e. for up to 5000 iterations simulated annealing and randomized hill climbing fall under 3 ms while computation time for genetic algorithm is around 35 ms even for 1000 iterations and exponentially increasing from there on.

Comparing both classification accuracy and computation times, simulated annealing is the best performing optimization algorithm for the neural network.

5.6 Analysis with Back Propagation

This section covers the comparison of optimization algorithmic configurations that performed best at correctly classifying the diabetic patients with the back-propagation based on neural network from assignment 1.

Accuracy for the 7-3-1 multi-layer perception using Weka was documented and comparison was done with the optimization algorithm performance which is shown in Figure 13 [LEFT]. It is clearly evident that back-propagation has best performance in terms of accuracy for both the training and the test set.

Figure 13 [RIGHT] (excluding computation time for genetic algorithm as it takes a lot of time) shows that back-propagation beats the computation times for both simulated annealing and randomized hill climbing by a considerable margin.

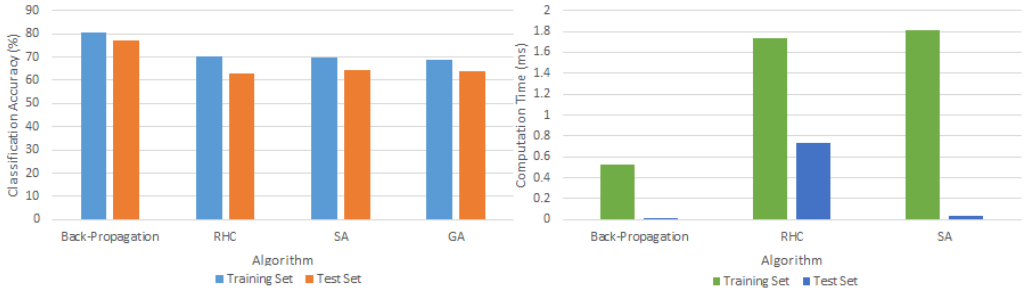

Figure 13: [Simulated Annealing] - LEFT: Correct Labels (Training/Test set), RIGHT: Computation Time (Training/Test set)

Considering that I split the data 70/30 for training and test set, simulated annealing performed relatively well among the optimization algorithms but could not out perform back-propagation.

6 Conclusion

The assignment was a mix of a lot of aspects of randomized optimization from evaluating performance against optimization problems to finding optimal neural network weights.

The analysis of all 4 algorithms stretched my understanding of the optimization algorithms highlighting their use pertaining to machine learning. It was noted that not all 4 optimization algorithms perform as well on all optimization problems covered in the assignment. Variants of these algorithms were deployed to obtain better results comparing and highlighting certain features of individual optimization algorithms.

The neural network analysis performed in this report is one of many ways to compare these optimization algorithms. I used classification accuracy as the medium for comparison in order to get a quick idea about the direction I should take to find optimal neural network weights and improve performance. It turned out, back-propagation deployed in assignment 1 surpassed the performance all 3 optimization algorithms with simulated annealing coming slightly closer.