Randomized Optimization

Gary Saavedra

gsaavedra3@gatech.edu

Introduction

This assignment was done using Pushkar’s ABAGAIL code.

Training a Neural Net

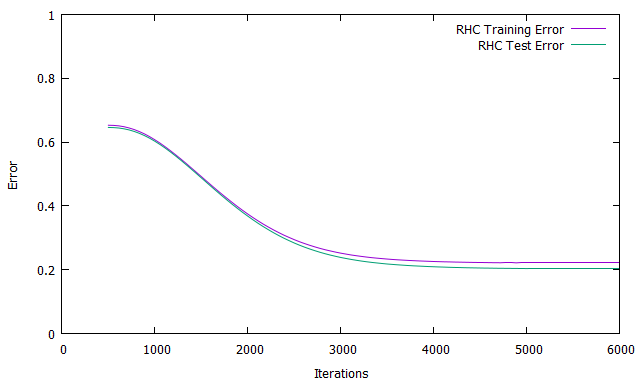
For the neural net 27 input nodes were used at the input layer, 7 nodes at the hidden layer, and 1 node at the output layer. Seven nodes were used at the hidden layer because that is the number at which the original neural net performed best. The neural net was trained using Random Hill Climbing (RHC), Simulated Annealing (SA), and a Genetic Algorithm (GA)(which one, parameters?) instead of BackPropagation. The training error is used as a fitness function (how to word this, since we are trying to optimize fitness not minimize cost). Sum of squared errors is used as the error measurement.

Random Hill Climbing

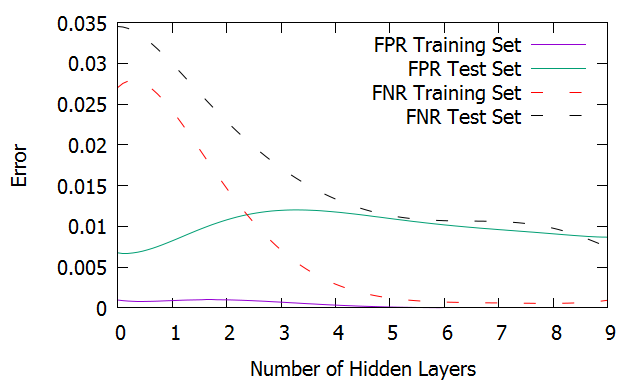
Figure 1 shows a graph of error vs iterations for a RHC algorithm. In the figure we can that both the training error and testing error start at about 65% for 1000 iterations. After about 4000 iterations the error bottoms out at 21%. The RHC algorithm is not able to choose a set of weights that improve the error rate beyond 21%. The RHC is most likely not doing well because it is getting stuck in a local optima. The error stays at .221 even after 1,000,000 iterations. If we look at the error from Assignment 1 using this same network we get a sum of squared errors of 16%.

There are a few reasons we may be getting an error higher than our original. The global optima may be very narrow and hard to reach. This is known as a basin of attraction. Even though RHC has random restarts and therefore many chances to converge to the global optimum, the large basin of attraction makes it unlikely the global optimum will ever be found. The algorithms may also not generalize a continuous space perfectly.

Strange part of the graph. The training error is lower than the testing error. However, this is not a major deal since they are very close together that means they are within the range of variance for each other. The training and testing error closely matches in the graph.



*Figure 1: Error vs Iterations for RHC*



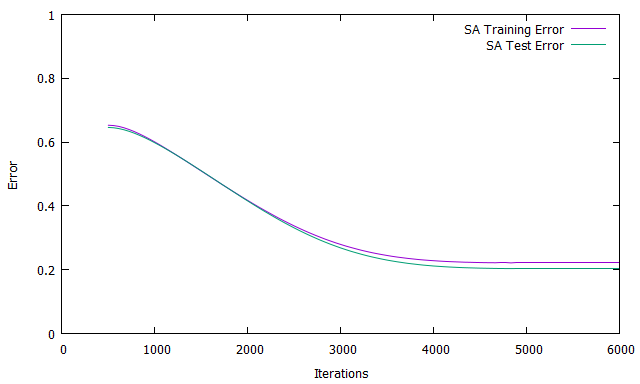
*Figure 2: Error rate for neural net using BackPropogation*

Simulated Annealing

For the SA algorithm the temperature starts at 10^11 and steps down 5% every iteration. The error stays at 21% even after 1,000,000 iterations.

Figure 3 shows the error vs number of iterations for the SA algorithm. This graph shows a similar trend to the RHC graph. At 1000 iterations the error is about 60%. After 6000 iterations the error has dropped to 21% error similar to the RHC algorithm. Why doesn’t SA converge after fewer iterations than RHC. It seems like it would since it’s less likely to get stuck at local optima.

It is possible there are many local optima with a weight configuration that gives approximately 21% error. These many local optima would dominate the probability function that decides which optima is chosen. This is probably not the case though since every algorithm is bottoming out at 21%. It is likely that this is the global optima and there is something else hindering the algorithms from reaching 16%.

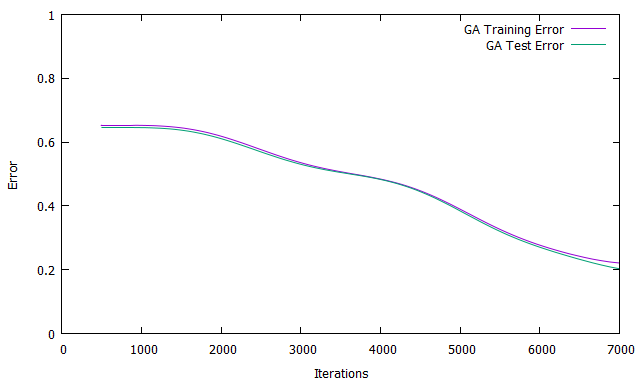


*Figure 3: Error vs Iterations for SA*

Genetic Algorithm

For the GA algorithm the parameters are as follows. The initial population size is 200. 100 members are chosen to mate and 10 members are chosen for random mutation. Is crossover used and how are the local bits chosen?

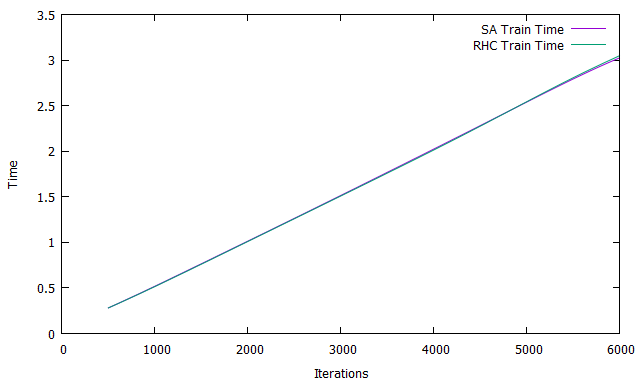
The error vs training iterations plot is shown in Figure 4. This plot differs significantly from the RHC and SA algorithms. The error also starts out at 65% but takes much longer to converge to the optimum of 21% error. It does not reach 21% until about 7000 iterations. We also see a bit of error increase around 4000-5000 iterations. This means even though the GA has had more training opportunities it actually had a larger error. This may be due to the randomness of the algorithm itself. The randomness of mutations and matings may cause this.



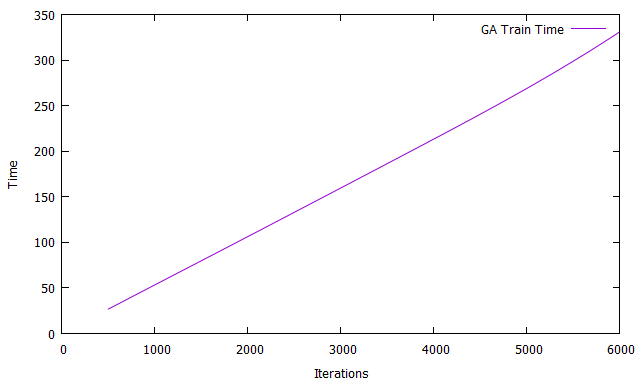
*Figure 4: Error vs Iterations for GA*

**Performance Comparison**

There are two figures showing the training time versus the number of iterations used for each algorithm. Figure 5 shows the training times for SA and RHC. Figure 6 shows the training times for the GA. As the figures show there is significant difference in training times between the GA and the other algorithms. The RHC and SA take only about 3 seconds at 6000 iterations. Their lines are overlapping. The GA takes about 325 seconds at 6000 iterations, an increase by a factor of 10. Both graphs show a linear increase in training times.



*Figure 5: Training time for SA and RHC in seconds*



*Figure 6: Training Time for GA in seconds*

**Optimization**

How is time being measured? It looks like in the n queens alg he is not measuring the time for the train() method. What is the train method exactly? Why aren’t we including it?

Including the train() method does seem to make a difference for traveling salesman. For RHC instead of 1ms I got 230 ms when including train(). For MIMIC I got 22000 instead of 3 when including training time. Maybe pushkar made a mistake.

For max K coloring pushkar does include the training time.

Idea for graph. Results for each alg over ten different training sessions so we can remove some of the randomness.

N queens demonstrates speed and minimal number of moves for SA. GA performs slowly and doesn’t do much better. MIMIC performs really slow and doesn’t do much better. Does this mean cost of evaluating function is low.

Traveling salesman demonstrates MIMIC finding best solution for MIMIC but slowest time. GA also performs with a much slower time than n queens problem.

Need to find problem in which mimic performs quickly. Or possibly find a problem in which GA performs better than others or more quickly. MIMIC performs quickly on k color problem and seems to find good solution. SA fails to find solution. GA finds solution in less time though. For n =10000 mimic does better on time. May be a bug in k coloring check piazza favorites.

Variable ef in each test represents fitness function for each test. I need to find out what each fitness fnctions is for each test. What are we optimizing.

**N-Queens Problem**

What’s meant by number of moves in NQueensFitnessFunction. What’s the starting board? Is it random? The number of moves I roughly the same every time we run the test so I don’t think the starting board is random or it the final number of moves would vary more.

The final board configuration changes everytime we run the program. This is because there is random starting point for each algorithm as well as a random sequence of events.

I believe a single move can be considered a single function evaluation

Highlights the advantages of SA.

**RHC**

Time consistently is 1 ms. Make graph for different N

Takes more moves than GA and MIMIC but way less time.

How does number of moves change with increasing

large increase for all algs in number of moves vs N. For example N=100 moves is about 4800. For N=200 its about 18000.

**Simulated Annealing**

Time consistently is 1 ms. Make graph for different N.

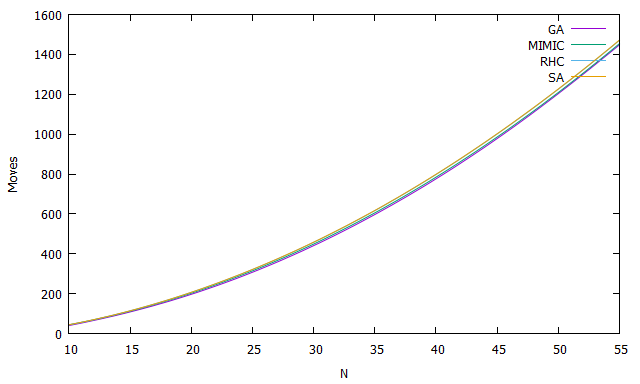
Takes more moves than GA and MIMIC but way less time.

**Genetic Algorithm**

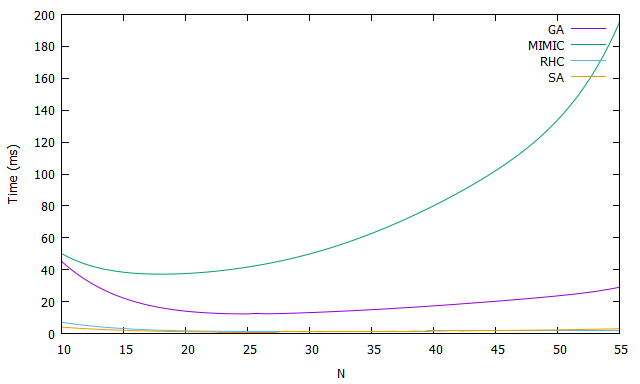
Time vs. N graph

**MIMIC**

Time vs. N graph, seems to be exponential increase of time vs. N.



*Figure 7: Number of Moves vs. N for N-Queens Problem*



*Figure 8: Time vs N for N-Queens problem*

**Four Peaks Problem**

The four peaks problem requires a maximization of a bit function. There are two ways to earn a reward. The first instance you will receive a reward of 100 if the number of leading 1’s and trailing 0’s are each greater than some T value. In addition you receive another award that is equal to your number of zeros or ones, whichever is greater. It gets its name from its four optimal peaks. There are two global optima and two local optima. The local optima are the cases when the search algorithm inputs all 1’s or 0’s resulting in a reward of N.

For our setup we use T = 11 and vary the number of bits, N, between 80-120. The figures are shown in figure 9 and 10. Figure 9 shows the max reward values obtained by each algorithm with N varying from 80 to 120. For each value of N tested the reward values were averaged over 10 trials in order to remove variance. Figure 10 shows the average time taken by each algorithm to compute its max reward value for different values of N. Once again the time values were averaged to remove variance. This problem will highlight the advantages of GAs.

**RHC**

RHC often gets stuck at one of the two local optima. This can be shown in figure 9. Following the curve for RHC we can see that the max value is always roughly equal to N. RHC has trouble escaping these local optima. From the RHC point of view every time it adds another 1 it increases its fitness. For instance, after creating 11 leading 1’s there are several neighbors that will increases fitness but many of them just involve adding another 1. From the RHC point of view adding a 1 to the tail increases fitness. It has no way of knowing that by keeping the last 11 bits at 0 it will incur a large maximum reward in the future. This weakness stems from the fact that RHC is an exploit algorithm. Without an ability to explore it has no way of finding the larger future reward. Once RHC adds a 1 to one of the last 11 bits of the tail it can no longer get the extra reward of 100. In order for the RHC to incur the large reward it must luck out and not add a 1 to the last 11 bits. The larger our value of T the larger the basin of attraction around the local optima and thus our RHC algorithm will have an even harder time.

**Simulated Annealing**

In Figure 9 we can see that SA performs somewhat better than RHC. Its ability to explore allows it to escape the local optima on occasion. SA still suffers often, however, from local optima. This is because the global optima is narrow. The SA algorithm still requires a lot of luck to happen into the global optima.

Figure 10 shows that the SA is able to perform quickly.

**Genetic Algorithm**

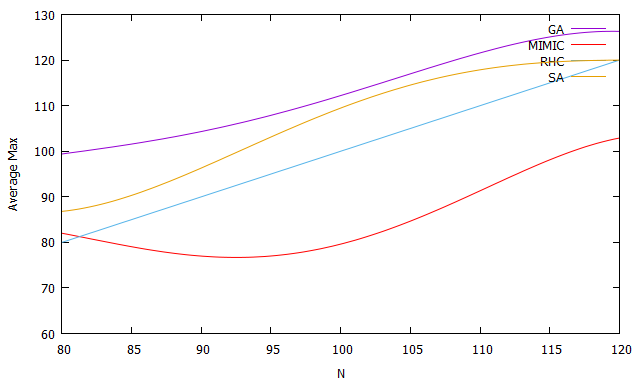
For the GA an initial population of 500 was used. Larger populations tended to do better. This is because with a larger population you are more likely to have individuals with many leading 1’s or many trailing 0’s. The amount of the population used for crossover was 400 and the amount used for mutation was 3. Single point crossover was used as this tends to give the best results for a four peaks problem. It gives best results because it is more likely that the two ends of the bit string will be combined without modification. This will lead to children which will obtain the reward of 100 that can continue to increase their fitness.

If we look at Figure 9 we can see that the GA outperforms all other algorithms. While the RHC and SA tend to get stuck at local optima the GA averages a performance above it. Figure 10 also shows that GA is able to achieve this performance in a very short amount of time rivaling both the SA and RHC. The time required for the GA does not increase significantly for increasing N.

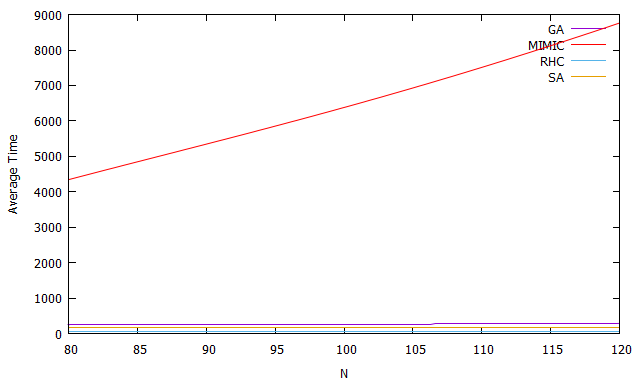
**MIMIC**

Figure 9 shows that MIMIC is the worst performer on the four peaks problem.

Figure 10 also shows the time requirement for MIMIC is much higher than the other algorithms. We can also see that time needed increases linearly with increasing N. MIMIC’s attempt to keep track of structure in this case is overkill.



*Figure 9: Average Max vs N for Four Peaks,*



*Figure 10: Average Time vs N for Four Peaks*

**Max K-color Problem**

This version of the Max K-Color problem uses the algorithm to find how many iterations it takes to color the graph appropriately. If it cannot be k-colored then the algorithm returns the number of iterations it takes to find out the graph is not k-colorable.

It seems that RHC,SA, and GA often fail. MIMIC will succeed a few times and then fail everytime after. Why does it continue to fail after a certain K has been reached? Is the graph not k colorable after that point? Why are the other algs failing so often. It seems that N must be really small for them to succeed.

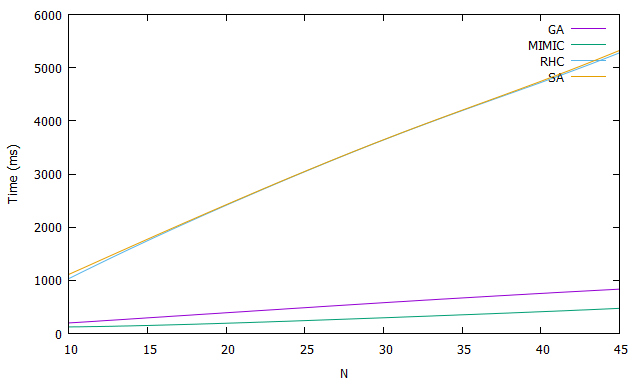
For N = 100,1000 MIMIC succeeds 3/9 times. The success probably depends on K as well. This if for L=3. For L=1 MIMIC fails to find a solution which is odd because it is just a chain. As long N is divisible by K there should be a chain that works. It’s strange that MIMIC can’t find this. Remember video quiz from lecture. MIMIC may be having trouble finding the chain distribution.

The function being maximized is the number of adjacent nodes with different colors.

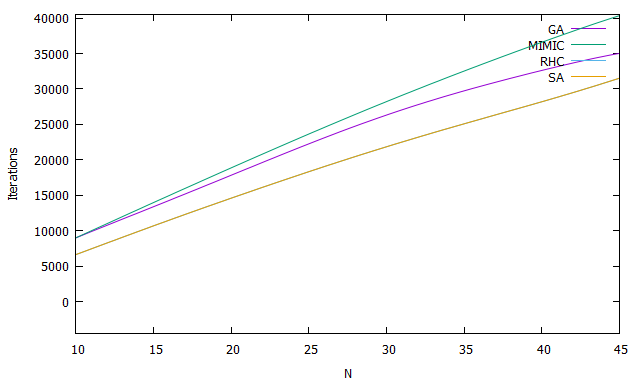
Highlights advantages of MIMIC.

The graphs are for L=3, N=1000

MIMIC succeeds for first 4 of 9. GA succeeds for 1st. SA and RHC fail for all



*Figure 11: Time vs N for Max K-Coloring problem*



*Figure 12: Iterations vs N for Max K coloring*