Randomized Optimization

Gary Saavedra

gsaavedra3@gatech.edu

Introduction

This assignment was done using Pushkar’s ABAGAIL code.

Training a Neural Net

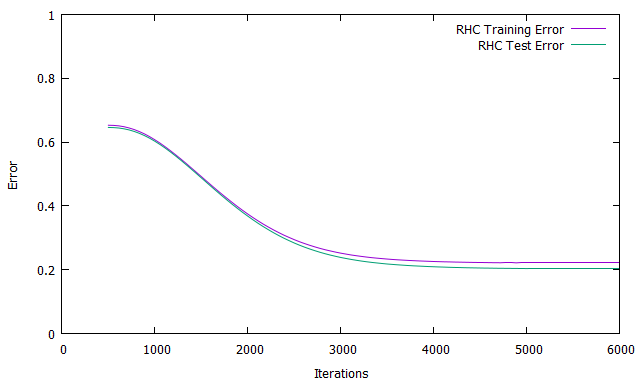
For the neural net 27 input nodes were used at the input layer, 7 nodes at the hidden layer, and 1 node at the output layer. Seven nodes were used at the hidden layer because that is the number at which the original neural net performed best. The neural net was trained using Random Hill Climbing (RHC), 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

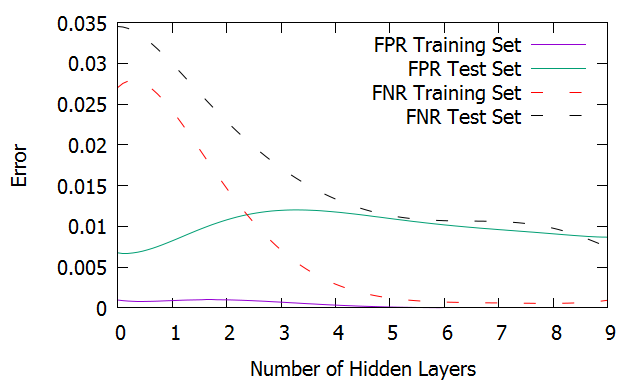
Figure 1 shows a graph of error vs iterations for a RHC algorithm. In the figure we can that both the training error and testing error start at about 65% for 1000 iterations. After about 4000 iterations the error bottoms out at 21%. The RHC algorithm is not able to choose a set of weights that improve 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*



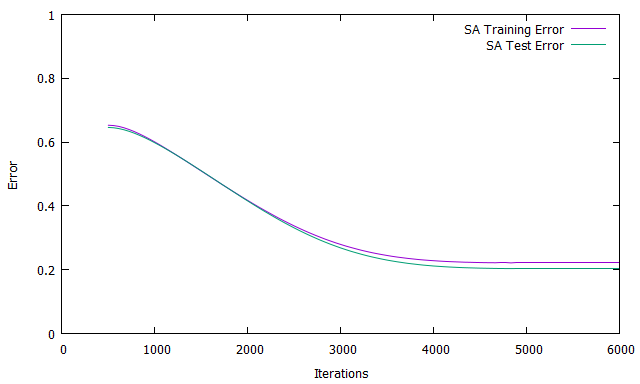
*Figure 2: Error rate for neural net using BackPropogation*

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%.

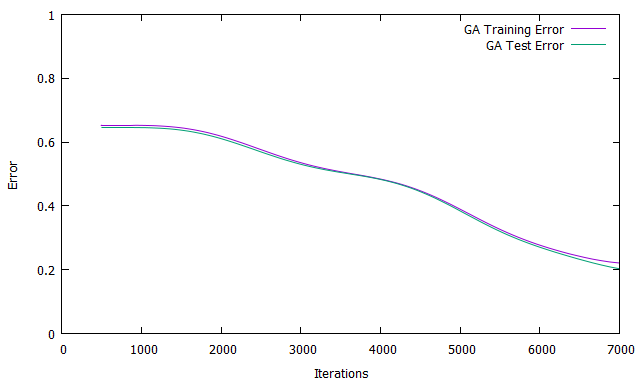


*Figure 3: Error vs Iterations for SA*

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?

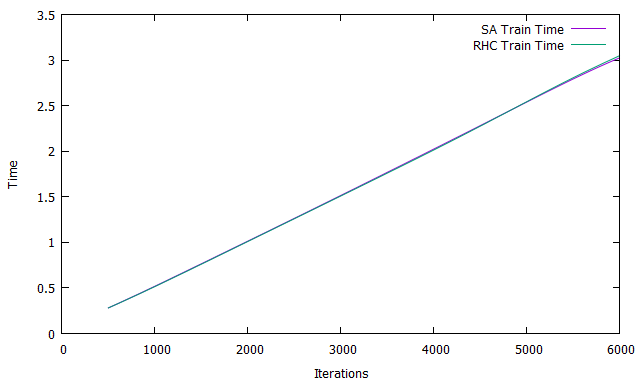
The error vs training iterations plot is shown in Figure 4. This plot differs significantly from the RHC and SA algorithms. The error also starts out at 65% but takes much longer to converge to the optimum of 21% error. It does not reach 21% until about 7000 iterations. We also see a bit of error increase 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.



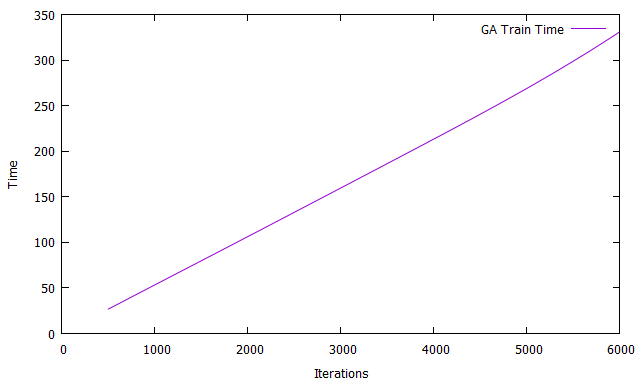
*Figure 4: Error vs Iterations for GA*

**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**

The N-Queens problem is a well known computer science problem. It involves placing N queens on a N x N chessboard such that no queen attacks another. The algorithm used is from the ABAGAIL library. The fitness function is attempting to maximize non-attacking pairs of queens.

The max fitness obtained by each algorithm for varying board sizes, N, is shown in Figure 7. Figure 8 shows the time taken by each algorithm to maximize fitness for varying N.

**Why is it Interesting?**

The N-Queens problem is an NP-hard problem. It takes polynomial time to find a single solution and exponential time to find all solutions for a given N x N board. For the case of an 8 x 8 board there are 4,426,165,368 possible board configurations but only 92 solutions. In general there are N^N possible boards. With such a large search space a simple brute-force algorithm will not suffice to find a solution in a reasonable amount of time.

The N-Queens problem will highlight the advantages of SA and RHC. It will also show the weaknesses of MIMIC and GA.

**RHC**

The algorithm starts by generating a random board configuration. From there it considers moving a single queen and all the possible moves that can be taken. It compares the fitness of each of these moves and takes the one with the highest fitness. If it gets stuck in a local optima it restarts with another random board configuration.

The RHC uses 100 iterations.

As can be seen in Figure 7 all the algorithms have a comparable performance. By looking at Figure 8 we can see that RHC outperforms the GA and MIMIC. For N within the range of 10-55 the RHC takes only about 1 ms.

The N-Queen fitness search space lends itself well to the RHC algorithm.

**Simulated Annealing**

For the SA algorithm a temperature of 10^11 and a cooling factor of 0.1 were used. These parameters gave the best performance out of SA. The SA uses 100 iterations.

The SA algorithm for the N-Queens problem works similar to the RHC . However, it is given a parameter temperature which allows the SA with some probability to choose a neighbor with lower fitness. This parameter allows the SA algorithm to escape local optima that RHC would get stuck in.

Looking at Figure 7 and 8 we can see the SA algorithm’s performance is comparable to the RHC. For the same reasons as the RHC the SA performs well. SA’s time performance is slightly better than RHC. This is because of its ability to explore. The high starting temperature allows SA to explore and is less likely to get stuck in a local optima. Thus it is less likely there will be a need for a random restart and having to start the search process over again. However, the global optima is still unlikely to be found in only a few attempts, thus the time is only slightly better.

**Genetic Algorithm**

For the GA algorithm an initial population of 200 is used. The number of the population selected for mutation is 10. There is no crossover used. The number of iterations is 100. These parameters gave the best performance. Not using crossover makes sense for this problem. There is no locality in this problem. Taking a half of two high fitness boards and combining them will often not yield a better board.

The GA algorithm for the N-Queens problem works by choosing a population of 200 random board configurations. From the population of board configurations it chooses 10 to randomly mutate. This process continues until the number of iterations is used up.

Figure 7 shows the GA’s performance in finding the optima is comparable to all the other algorithms. Figure 8, however, shows that GA suffers from long run times. Whereas the RHC and SA run times are about 1 ms, the GA’s run time is about 20 ms.

The GA’s poor performance is due to the structure of the problem. First of all crossover hurts performance for this problem, so we are limited to random mutations.

For each of the 200 board configurations we must evaluate the fitness function every iteration. This will already take 200 x more time than RHC. The advantage is that we have 200 starting boards to work with. However, there is not much gain. The 200 boards only make up a tiny fraction of the total board configurations. Any one of these 200 boards is unlikely to be close to the global optima.

It gets even worse. Even though we choosing the best boards to mutate, the mutations are random. In comparison to RHC which at least finds a neighbor with better performance we don’t get that guarantee with random mutation. Thus we must randomly stumble upon the solution.

Random mutations lead to a random convergence to the optima.

**MIMIC**

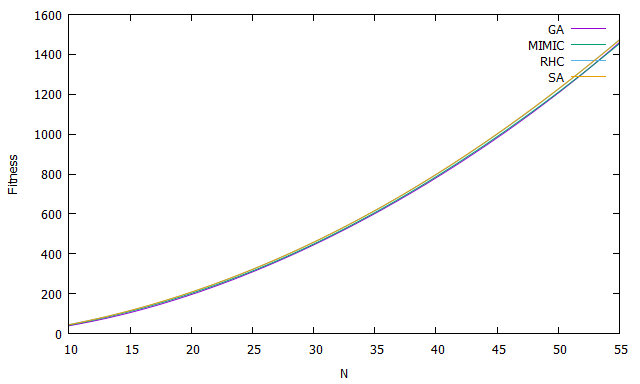
For the MIMIC algorithm we generate 200 samples per iteration and keep the best 10. The N-Queens problem needs only 5 iterations.

Figure 7 shows that MIMIC performs at the same level as the other algorithms when finding the max fitness. However, looking at Figure 8 we can see that MIMIC takes a significant amount of time to reach the same optima as the other algorithms. This is an indication that the cost evaluation the fitness function for the N-Queens problem is low. If it were high we would expect MIMIC to perform better than RHC and SA. This is because MIMIC many fewer iterations than RHC and SA. If the cost of evaluating the fitness function every iteration was high, SA and RHC would perform slowly.

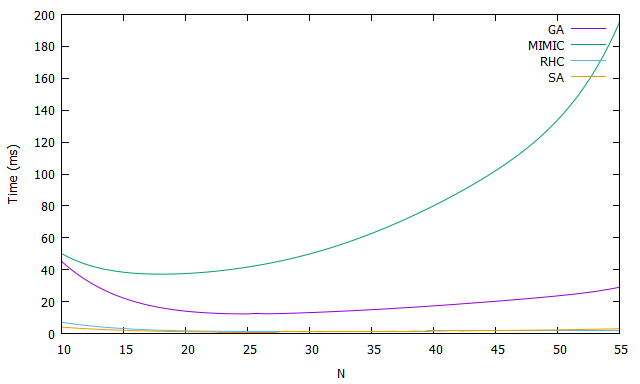
MIMIC performs more slowly because every iteration it must draw samples. From those samples it must find the most fit samples and then use those samples to create a new distribution. This process is time consuming and repeats every iteration.

MIMIC still finds a comparable optima to the other algorithms because there is an underlying structure to the problem.

MIMIC may perform better with a distribution other than dependency trees.



*Figure 7: Fitness 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.

**Why is it Interesting?**

The search space of the four peaks problem differs greatly from the other two selected problems. There are four optima. The two local optima have very wide basins of attraction. The path to the local optima cover a large portion of the search space. In contrast, the two global optima are very narrow. The path to their peaks covers a very narrow portion of the search space. This unique space is created by the large reward given for satisfying two conditions. As we will see this search space will illuminate the weaknesses of RHC, SA, and MIMIC while showing the strengths 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**

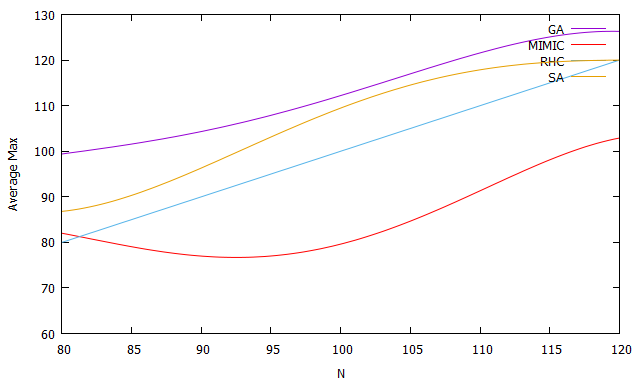
For the GA an initial population of 500 was used. Larger populations tended to do better. This is because with a larger population you are more likely to have individuals with many leading 1’s or many trailing 0’s. The amount of the population used for crossover was 400 and the amount used for mutation 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.

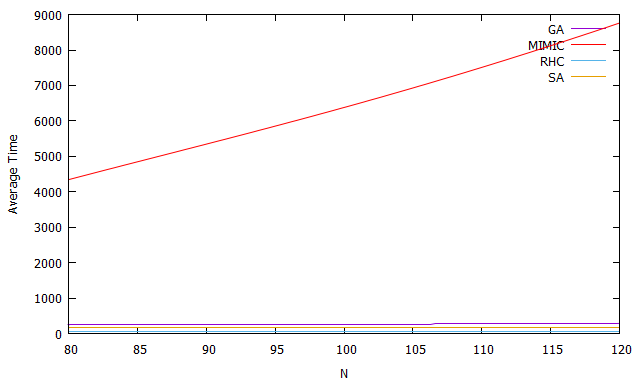
**MIMIC**

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**

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.

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

The Max K Color problem is another popular problem in Computer Science. It involves building a group of vertices each interconnected a number of times, L. There are N vertices. There are a certain number of colors, K, that each node can take on. The goal is to create a graph with each node a different color than adjacent nodes.

For our setup we use L=3 and N=1000. K varies from 10 to 45. Figure 11 shows the time taken by each algorithm to obtain an optimum. Figure 12 shows the maximum of distinct colored nodes sizes of K between 10 and 45.

**Why is it Interesting?**

The Max K Coloring problem provides an interesting search space for our algorithms.

The space proves difficult for GA, SA and RHC.

This problem highlights the advantages of MIMIC.

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

**Random Hill Climbing**

The RHC algorithm starts by generating a random graph of size 1000 with 3 edges between each vertex with K colors. From this starting point it finds all graphs with one color changed as its set of neighbors. It chooses the neighbor that provides the greatest improvement to the fitness function. If RHC hits a dead end it does a random restarts and starts with a new random graph. The number of iterations the RHC has to find an optimal function is 20,000.

The performance for RHC in Figure 12 is hard to see but it overlaps the SA curve. Figure 11 shows the time performance of RHC. We can see that RHC is one of the worst performers.

The fitness space is difficult for RHC to search. It is full of local optima. It is often the case that the RHC fails to find the global optima.

Will tend to get confused by the different optima and will get pulled in different directions.

**Simulated Annealing**

The initial temperature is 10^11 and the temperature decay factor is 0.1. Similar to the RHC 20,000 iterations were used.

Figures 11 and 12 show that SA’s performance is similar to RHC.

**Genetic Algorithm**

An initial population of 200 was used. 10 members were chosen for single point crossover and 60 were chosen for mutation. The GA uses 60 iterations.

In Figure 11 we can see that GA’s performance is much better than SA and RHC. Figure 12 shows that it also performs well when maximizing the fitness function.

The structure of the graph is well suited for GAs. The crossover function is able combine different portions of the graph that are performing well. Although this is not ideal it can still nonetheless create a graph with a high optima. If two portions of two separate graphs are performing well combining them will often lead to a higher optima even if it is not the global optima. Mutation will further help by adding an ability to generate more random graphs.

Why doesn’t it do as well as mimic.

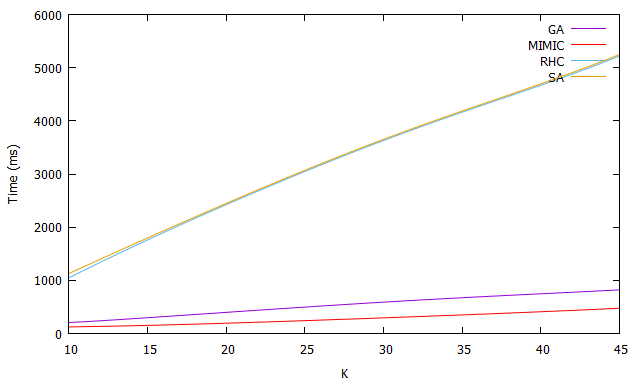
**MIMIC**

MIMIC takes 200 samples and keeps 100 of the fittest samples. It uses 5 iterations.

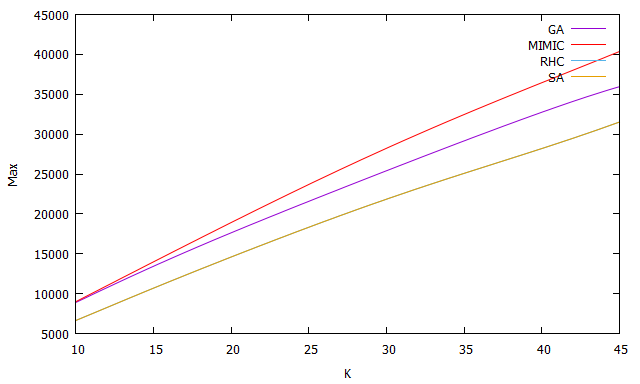
Figure 11 shows that MIMIC has the best time performance compared to the other algorithms. Figure 12 shows that MIMIC consistently maximizes the fitness function.

The improved time performance, along with MIMIC only taking 5 iterations shows that the cost of evaluating the fitness function is high. To evaluate the fitness function we must evaluate each vertex of the graph and assure that none of the connected vertices have the same color.

MIMIC does well on the K-color graph because of the dependency tree used for the underlying probability distribution.



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



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