Randomized Optimization

Gary Saavedra

gsaavedra3@gatech.edu

Introduction

This assignment was done using Pushkar’s ABAGAIL code.

Training a Neural Net

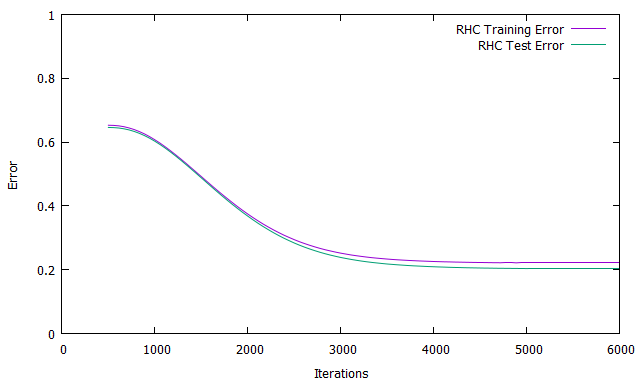
For the neural net 27 input nodes were used at the input layer, 7 nodes at the hidden layer, and 1 node at the output layer. Seven nodes were used at the hidden layer because that is the number at which the original neural net performed best. The neural net was trained using Random Hill Climbing (RHC),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

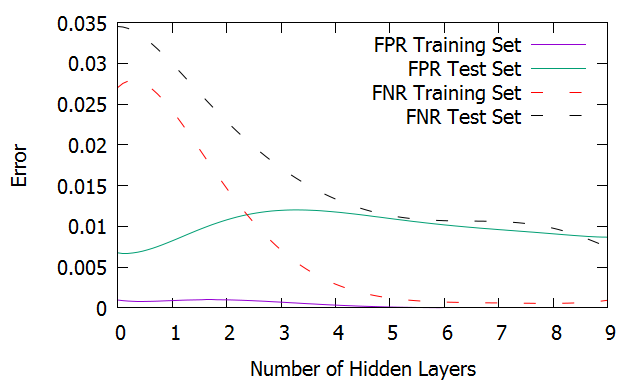
Figure 1 shows a graph of error vs iterations for a RHC algorithm. In the figure we can that both the training error and testing error start at about 65% for 1000 iterations. After about 4000 iterations the error bottoms out at 21%. The RHC algorithm is not able to choose a set of weights that improve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



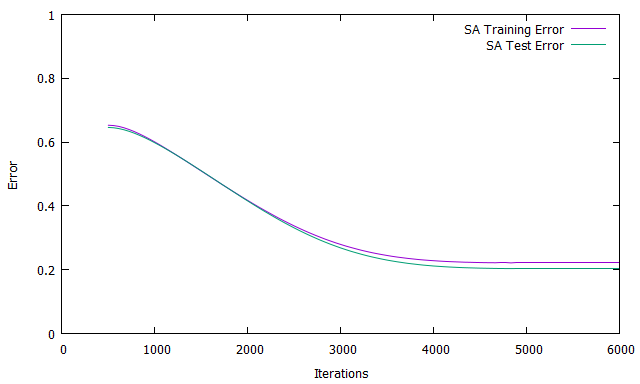
*Figure 2: Error rate for neural net using BackPropogation*

Simulated Annealing

For the SA algorithm the temperature starts at 1011 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%.

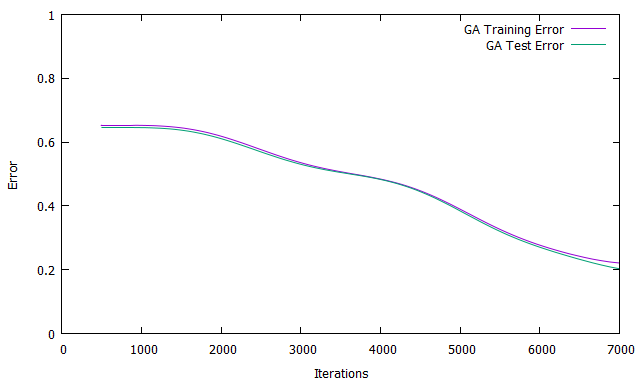


*Figure 3: Error vs Iterations for SA*

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?

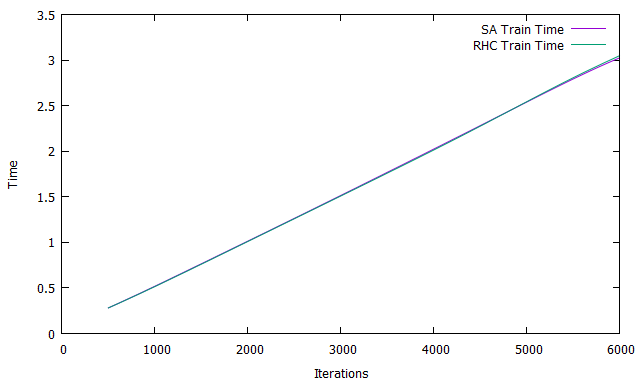
The error vs training iterations plot is shown in Figure 4. This plot differs significantly from the RHC and SA algorithms. The error also starts out at 65% but takes much longer to converge to the optimum of 21% error. It does not reach 21% until about 7000 iterations. We also see a bit of error increase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



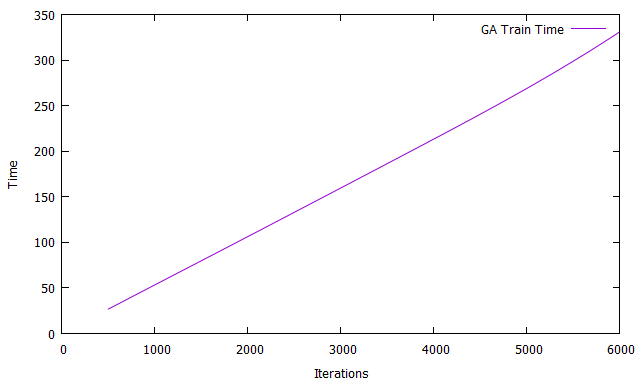
*Figure 4: Error vs Iterations for GA*

**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**

Optimization involves maximizing a particular function called the fitness function. The fitness function we are attempting to maximize varies for each particular problem. To demonstrate optimization three well known problems, the N-Queens, Four Peaks, and Max K-Coloring problems were chosen. These three problems demonstrate the strengths and weaknesses of RHC, SA, GA, and MIMIC. These problems are also illustrative of the No Free Lunch Theorem which states that if an algorithm performs well on a particular problem then it must perform poorly on another problem.

**N-Queens Problem**

The N-Queens problem 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. The number of possible board configurations increases exponentially with increasing N. 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 RHC algorithm starts by generating a random board configuration with all N queens placed in random positions on the board. From there it chooses a single queen and considers 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 optimum it restarts with another random board configuration and repeats the process. The only parameter that RHC can vary is the number of iterations. For the graphs shown the RHC uses 100 iterations. This setting gave the best run times while also finding optimal solutions.

As can be seen in Figure 7 all the algorithms have a comparable performance when finding the maximum fitness. By looking at Figure 8, however, 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. In contrast, the GA takes an average of 20 ms and the time taken to train MIMIC increases linearly, with N with the worst case being about 200 ms.

The N-Queen fitness search space lends itself well to the RHC algorithm. When the RHC is choosing between the moves a particular queen can take, it is likely that at least one of those moves will increase fitness. If it happens that there are no moves that increase fitness the RHC can start with a new board configuration. It is unlikely after 100 iterations that one of the global optima was not reached.

**Simulated Annealing**

For the SA algorithm a temperature of 1011 and a cooling factor of 0.1 were used. The SA algorithm, similar to RHC, also uses 100 iterations. These parameters gave the best performance out of SA.

The SA algorithm for the N-Queens problem works similar to the RHC algorithm. However, it is also given a temperature parameter which allows the SA algorithm to choose a neighbor with lower fitness with some probability. The temperature parameter allows the SA algorithm to explore the space by moving the queen to a non-optimal position. This allows the SA to escape local optima that the RHC would get stuck in and results in less random restarts to alternate board configurations.

Looking at Figure 7 and 8 we can see that SA’s performance is comparable to the RHC. SA performs well on the N-Queens problem for many of the same reason the RHC performs well. The search space is ideal for hill climbing algorithms. SA, however, has the additional advantage of temperature which allows it to escape local optima and converge to the global optima for quickly. For this reason SA’s time performance is slightly better than RHC. The high starting temperature allows SA to explore alternate board configurations early on that would not be considered by RHC. 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 with 200 iterations used for evolution. Each member of the population in this case is a random board configuration. From the population of board configurations the GA probabilistically chooses 10 of the fittest members to randomly mutate. This process continues for 200 iterations. Crossover tends to degrade performance for this problem, hence it is not used. The reason crossover tends to be detrimental to performance is the non-locality of parameters in this problem. In order for crossover to be effective we must assume locality when optimizing certain portions of our fitness space. In this case using single point crossover would take a piece from each of two well performing board configurations and combine them. However, taking a portion of two high fitness boards and combining them will often not yield a better board. The queens on one portion of the board are not independent of the queens on the other portion of a board. The combination of the two halves will likely result in a board with many attacking queens. For this reason we cannot assume locality.

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. One of the strengths of the GA is crossover. The ability to take two strong performers and combine their strongest parts is one the main advantages of the GA. However, in this problem crossover only hurts the GA because locality on portions of the hypotheses cannot be assumed. The inability to use crossover limits us to random mutations. Only using mutations leaves us no better off than an RHC algorithm. In reality, however, using only random mutations will cause the GA to perform worse than RHC. The RHC at least has the ability to choose a neighbor that increases fitness. A GA using only mutations takes no consideration of fitness and chooses a neighbor randomly.

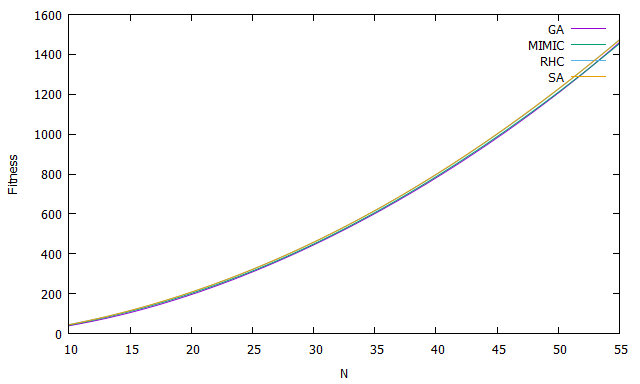
The increased time to find an optimum comes from the GA’s number of function evaluations. For every iteration we must evaluate the fitness function for each of the 200 board configurations in order to choose the most fit members. This process will require more time than the process the RHC and SA use to evaluate fitness. The advantage, of course, is that we have 200 starting boards to work with rather than 1, as is the case with RHC and SA. However, there is not much gain in this situation. 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 and will, thus, require a large number of mutations before finding the global optima.

**MIMIC**

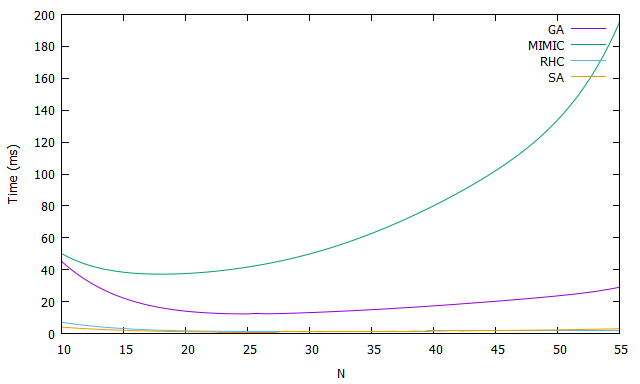
For the MIMIC algorithm we generate 200 samples per iteration and keep the best 10. These 10 samples are used to create a new distribution to sample from the next iteration. The N-Queens problem needs only 5 iterations to find a global optimum.

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 optimum compared to the other algorithms. The large amount of time needed is an indication that the cost of evaluating the fitness function for the N-Queens problem is low. If the cost of evaluating the fitness function is high we would expect MIMIC to perform better than RHC and SA. With only 5 iterations MIMIC performs very few fitness function evaluations. In contrast, RHC and SA use 100 iterations thereby performing many more function evaluations than MIMIC. If the cost of evaluating the fitness function every iteration was high, SA and RHC would perform slowly.

MIMIC’s slow performance is due to the number of steps taken during each iteration. For each iteration it draws 200 samples from the fitness function space. For each of those samples it must evaluate the fitness function to find the most fit samples. It then uses these samples to create a new distribution and repeat the process.



*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 takes a bit string of size N and maximizes a fitness function based on two conditions. For the first case you will receive a reward of 100 if the number of leading 1’s and trailing 0’s are each greater than some specified T value. In the second case you receive a reward that is equal to your number of zeros or ones, whichever is greater. The fitness function for the four peaks problem is the sum of the rewards obtained from these two cases.

For our setup we use T = 11 and vary the length of the bit string, N, from 80-120. Figure 9 shows the fitness function evaluations for 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 the max fitness value for different values of N. The time values were averaged over 10 trials to remove variance.

**Why is it Interesting?**

The search space of the Four Peaks problem differs greatly from the other two selected problems. In the four peaks problem there are four optima. There is a set of two local optima and two global optima. The local optima are the cases when the search algorithm inputs all 1’s or 0’s resulting in a reward of N. The global optima results when you earn the reward of 100 for meeting the head and tail conditions, as well as, the reward for maximizing the number of ones or zeros.

The two local optima have very wide basins of attraction. The paths to the local optima cover a large portion of the search space. In contrast, the two global optima are very narrow. The paths to their peaks cover a very narrow portion of the search space. This unique space is created by the large reward given for satisfying the single condition for the head and tail. As we will see this search space will illuminate the weaknesses of RHC, SA, and MIMIC while showing the strengths of GAs.

**RHC**

For the RHC there is only one parameter to vary in the ABAGAIL library, the number of iterations. In this case the number of iterations used was 200,000. Increasing beyond this number of iterations did not improve performance. The RHC algorithm begins by creating a random bit string of length N. It then flips bits from each position and performs a fitness function evaluation to choose which neighbor to move to.

RHC often gets stuck at one of the two local optima. This can be seen in Figure 9. Following the curve for RHC we can see that the max value is always roughly equal to N. From the RHC point of view every time it adds another 1 or 0 (whichever is currently the majority) it increases its fitness. For instance, after creating several 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 exploitation 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 get lucky and not add a 1 to the last 11 bits until the first 11 bits are all ones. 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**

For the ABAGAIL version of SA there are 3 parameters that can vary. The initial temperature is 1010 and the decay rate is .95. The number of iterations used is 200,000. These parameters gave the best results for SA.

In Figure 9 we can see that SA performs somewhat better than RHC. Its ability to explore allows it to occasionally escape the local optima that RHC gets stuck in. SA still suffers often, however, from the large basins of attraction of the local optima. Because the paths to the global optima are contained in a narrow part of the search space, the SA algorithm still requires a lot of luck to reach a global optimum. SA will perform better on search spaces where the global optima have large basins of attraction. While temperature is high the SA will overcome the local optima and search the part of the space containing the global optima. However, as temperature begins to cool the SA is less likely to explore and will converge towards the optima by following a path of neighbors that increase fitness. If the path to the global optima is narrow SA is unlikely to wander into the portion of the search space leading to global optima before the temperature cools.

Figure 10 shows, however, that the SA is able to perform quickly. Overall the SA algorithm performs well on this algorithm. Its performance rivals the GAs performance both in terms of maximized fitness and time.

**Genetic Algorithm**

The ABAGAIL version of the GA for this problem works by creating a population of different bit strings. For the GA an initial population of 500 was used. Larger populations tended to do better. This is because a larger population is more likely to contain 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 the Four Peaks problem. Single point crossover gives best results because it is more likely that the two bit strings will be combined without modification to either end. This will create populations where many members already contain the necessary bit pattern for a reward of 100. These populations can then be further improved and a reward greater than 100 will be attained.

If we look at Figure 9 we can see that the GA outperforms all other algorithms by obtaining a greater optimum than the other algorithms. While the RHC and SA tend to get stuck at local optima the GA is able to find optima greater than the local. The GA’s performance is due to the structure of the fitness search space. The GA will perform well with a fitness space whose structure contains local groupings that can be optimized separately. The bit pattern for the Four Peaks problem can be thought of as having different localized positions. The first 11 bits and last 11 bits each form a local grouping, while the bits in between these form another local grouping. The GA is able to improve fitness overall by improving the fitness of each of the individual local positions and then combining the results.

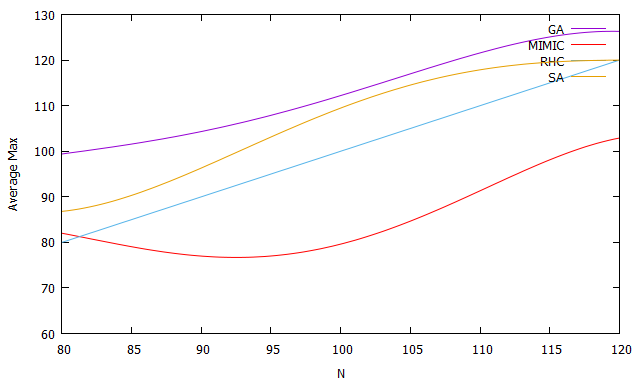
Figure 10 also shows that GA is able to obtain a large optimum in a very short amount of time rivaling both the SA and RHC. The excellent time performance of the GA is due to its selection process. With an initial population of 200 it is likely that there will be at least one bit string in the population with a large number of 1’s that are not in the final 11 positions or a large number of 0’s that are not in the first 11 positions. These individuals will have a high fitness and will be carried over to the new population. Even if the initial population does not contain these types of individuals the successive populations will be likely to contain them. The presence of the high fitness individuals in one of the early populations will cause the GA to quickly converge towards an optimum greater than a local optimum.

Figure 10 also shows the time required for the GA does not increase significantly for increasing N. This is because the execution time of the GA does not depend strongly on the size of the bit string. The GA obtains 200 individuals and performs crossover and mutation on them, but that is independent of the individual’s length. The fitness evaluation only depends on two checks of the bit string. It must check if the first 11 bits are one and the last 11 bits are zero. This process is also constant with changing N since T is unchanging. The only thing that increases the fitness evaluation time is counting the number of ones or zeros and adding them to the reward value. This increases with N but the cost is small.

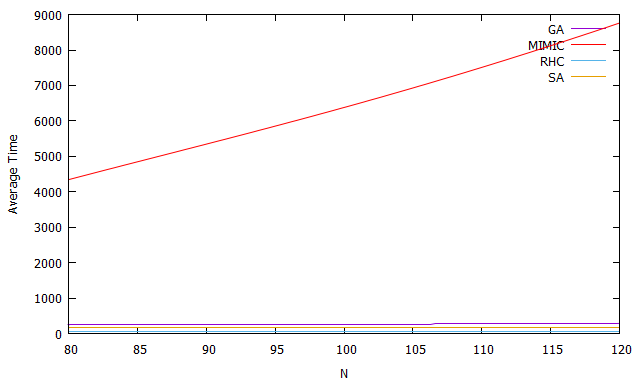
**MIMIC**

The MIMIC algorithm takes 200 samples and keeps the 20 with the highest fitness. It uses 1000 iterations. These parameters gave the best performance for MIMIC. The high number of iterations is needed to effectively search this space where the global maxima make up only a small portion of the search space.

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. The poor performance of MIMIC is due to the structure of the search space for the Four Peaks problem. When MIMIC takes its samples most likely they will be chosen from the area of the space containing the local optima. MIMIC will then use some portion of these samples to create another distribution. Since any of these samples are unlikely to be from the area of the global optima the new distribution will be biased towards the local optima. This process repeats until we either exhaust the allotted iterations or we return a distribution containing the local optima. Increasing the number of samples from 200 to 2000 greatly improves MIMIC’s performance on this problem. This is because with more samples we are more likely to pick points from the global optima for our distribution. However, we pay a cost in time with the average time increasing about 4x the original.



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



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

**Max K-Color Problem**

The Max K-Color problem involves building a group of vertices each interconnected a number of times, L. There are N vertices and a K amount of colors that each vertex can take on. The goal is to create a graph with each vertex having a different color than adjacent neighbor nodes.

For our setup we use L=3 and N=1000. K varies from 10 to 45 in the figures. Figure 11 shows the time taken by each algorithm to obtain an optimum. Figure 12 shows the fitness maximum vs K with K varying between 10 and 45. The function being maximized is the number of adjacent nodes with different colors.

**Why is it Interesting?**

The Max K Coloring problem is an NP-complete problem and provides an interesting search space for our algorithms. The space proves difficult for GA, SA and RHC while highlighting the advantages of MIMIC. In particular our MIMIC algorithm uses a dependency tree to represent the underlying structure. The underlying structure of the graph is very similar to the dependency tree.

**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 restart 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 because 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. RHC often fails to find the max K-Colorable graph because it tends to get confused by the different optima and will get pulled in different directions. When it eventually settles it has usually ended up at a local optima. The RHC will then restart but there are so many peaks that it is unlikely it will find the global optima.

Figure 11 shows the time performance of RHC is poor compared to GA and MIMIC. The poor performance is due to the high cost of evaluating the fitness function. For each K-colorable graph there are many neighbors and evaluating their fitness function involves comparing the colors of each of the nodes.

**Simulated Annealing**

The initial temperature is 1011 and the temperature decay factor is 0.1. Similar to the RHC, 20,000 iterations were used. The SA algorithm in ABAGAIL for this problem works similar to the RHC algorithm except the temperature parameter is added.

Figures 11 and 12 show that SA’s performance is similar to RHC. SA suffers from many of the same issues as RHC in this problem. In general, a hill climbing algorithm will have trouble with this space because of the many peaks. SA’s ability to explore in this case does not provide much benefit because there are many small peaks in the search space. During the high temperature phase SA is able to overcome some local optima. However, it is still unlikely to reach the global optima by the time the temperature has cooled. In addition the cost of evaluating the fitness function remains high, and thus gives the SA a poor performance time.

**Genetic Algorithm**

The GA works by creating an initial population of graphs with random color arrangements. In this case an initial population of 200 was used. Only 10 members were chosen for single point crossover and 60 were chosen for mutation. The GA for this problem uses 60 iterations. These parameters gave the best output for the GA.

In Figure 11 we can see that GA’s time performance is much better than SA and RHC. The time performance for the GA is better than SA and RHC because of the reduced amount of fitness function evaluations. The GA only needs to evaluate the fitness of the 200 members of the population. RHC and SA, on the other hand, must evaluate the fitness for each individual neighbor. A neighbor in this case is a graph with one color changed. That means for N=1000 there are 1000 different neighbors to evaluate for the RHC and SA algorithm. In addition the GA only uses 60 iterations.

Figure 12 shows that the GA also performs well when maximizing the fitness function. The GA is able to outperform SA and RHC due to its ability to take advantage of the structure of the K-color graph. The structure of the K-color 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 optimum because locality of different portions of the graph can mostly be assumed. For example if we use single point crossover and combine two halves of two graphs that are performing well we can expect the new graph to perform well. We can expect improved performance because only a few fitness points are lost at the joint where the two halves are linked if the linked nodes share the same color. The rest of the graph will perform independently of the linking point. Mutation can further improve performance by possibly mutating the color where the two halves are joined.

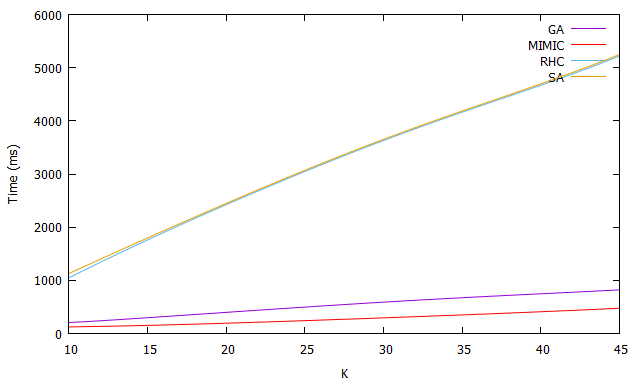
**MIMIC**

The MIMIC algorithm was obtained from the ABAGAIL library. It works by considering the space of K-color graphs and successively choosing stronger performers each iteration. For each iteration MIMIC takes 200 samples and keeps 100 of the fittest samples. In this case 5 iterations were used. These parameters gave the best performance.

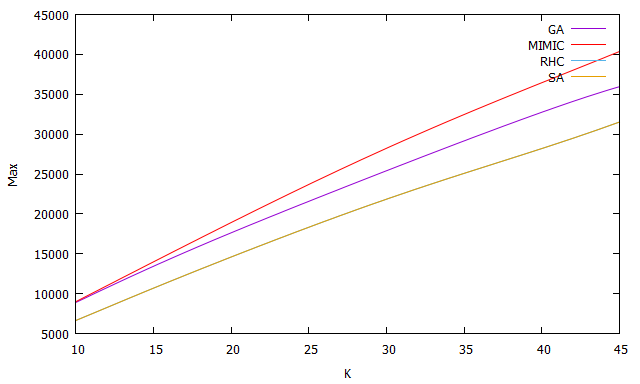
Figure 11 shows that MIMIC has the best time performance compared to the other algorithms. The improved time performance, along with MIMIC only taking 5 iterations to perform better than the other algorithms shows that the cost of evaluating the fitness function is high. To evaluate the fitness function we must evaluate each vertex of the graph and assure that none of the connected vertices have the same color. This means there are N function evaluations for each iteration. MIMIC’s excellent time performance is due to the fact that it uses so few iterations. RHC and SA, on the other hand, must make the function evaluation N times for every one of their 20,000 iterations.

Figure 12 shows that MIMIC consistently outperforms the other algorithms when maximizing the fitness function. The search space lends itself well to MIMIC. The curve leading up to the maximum fitness is wide and covers a large portion of the search space. Even though there are many small peaks in the path to the max, MIMIC’s sampling will be able to successively overcome these local peaks. It can do this because samples from the newest distribution are likely to contain improved fitness, in contrast to the Four Peaks problem where the odds of picking samples from the paths leading to the global optima were low.

Figure 11 and 12 show that MIMIC outperforms the GA. MIMIC is able to bypass the limitations of the GA by taking into account the global structure rather than local parts. GA assumed that certain portions of the K-color graph were independent when in fact they were not. This led to loss of fitness at joint points. MIMIC makes no assumption of locality and instead considers the structure as a whole.



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



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