Kaley Findley
CS 4641
3/12/17

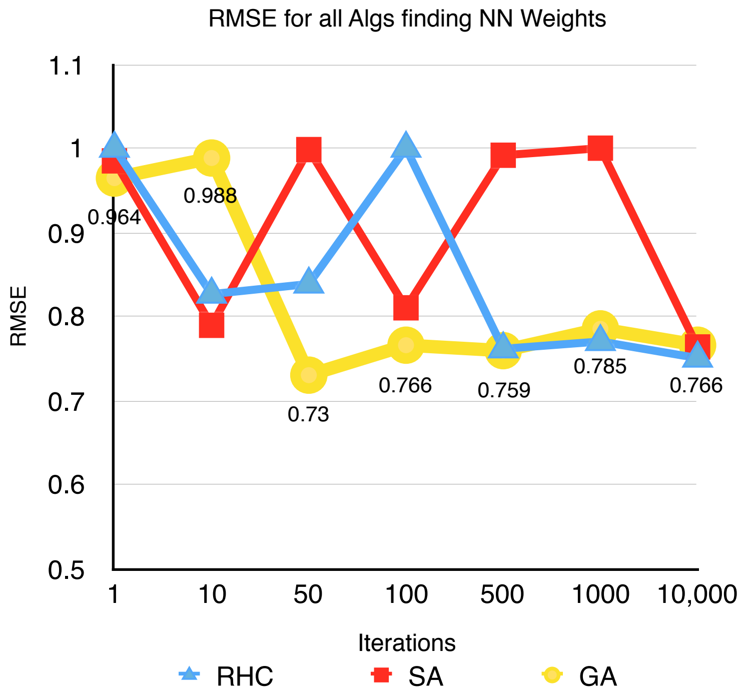
Randomized Optimization

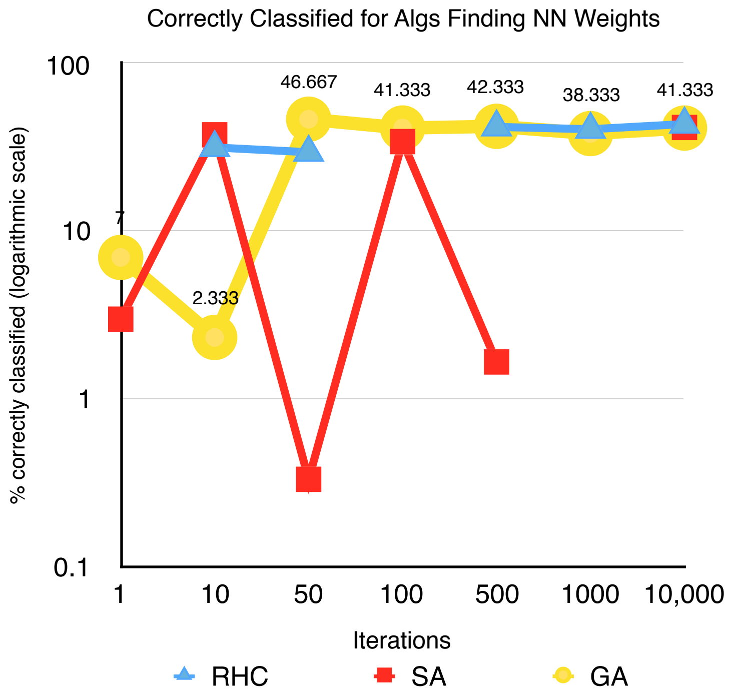
Introduction

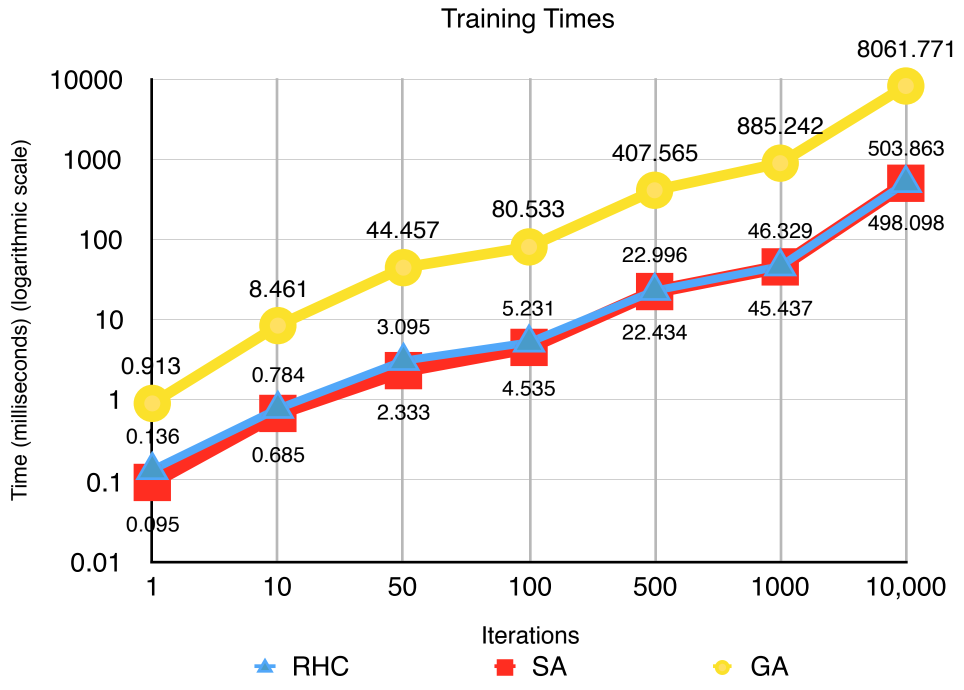
This assignment focuses on the advantages and disadvantages of various optimization search algorithms: Randomized Hill Climbing (RHC), Simulated Annealing (SA), Genetic Algorithm (GA), and MIMIC. This analysis will discuss the effects of running the first three algorithms through a Neural Network without back propagation as well as exploring what kinds of problems these algorithms excel (or fail) in. I will test the performance of RHC, SA, GA, and MIMIC in three optimization problems: Four Peaks, Max K-Coloring, and Traveling Salesman. I tested the effects on performance by manipulating the different parameters. For RHC I tried two different neighbor functions, DiscreteChangeOneNeighbor and ContinuousAddOneNeighbor and gradually increased the number of iterations. For SA, I manipulated the initial temperature and cooling exponent. The initial temperature illustrates how "hot" the algorithm starts as, which determines its ability to traverse large or small valleys in the data. The cooling exponent is simply how quick you want to "cool" or decrease that temperature. For GA, I changed the population size and the number of mutations, matings, and iterations. The parameter toMate determines how many individuals will be mated with one another and the toMutate parameter determines how many will be mutated. Lastly, for MIMIC, I changed the parameters for the number of samples and the number of samples to keep in the dependency for the next distribution estimate. I represented the results of the optimization problems by averaging the results of each parameter I changed. Due to the large amounts of data given in each test run, this was the most concise way to illustrate the effects.

Part 1: Neural Network

For the Neural Network part of the assignment, I passed in RHC, GA, and SA, to find the optimal weights for the network, and compared these results to the back propagation network for my wine quality data set from the first assignment. I ran first neural network with varying iteration values (1, 10, 50, 100, 1000, 10000) to see if that had a significant effect on how the optimization algorithms found the optimal weights. I used my largest set of my wine data; it had 3000 instances in it. I then compared the results of that neural network with the "best" neural net I found in the first assignment. I compared these two neural nets in terms of the lowest root mean squared error (RMSE). The first 3 graphs are for the neural net with the optimization algorithms. They are to illustrate the performance of RHC, GA, and SA in terms of reducing RMSE, correctly classified instances, and overall computation time.



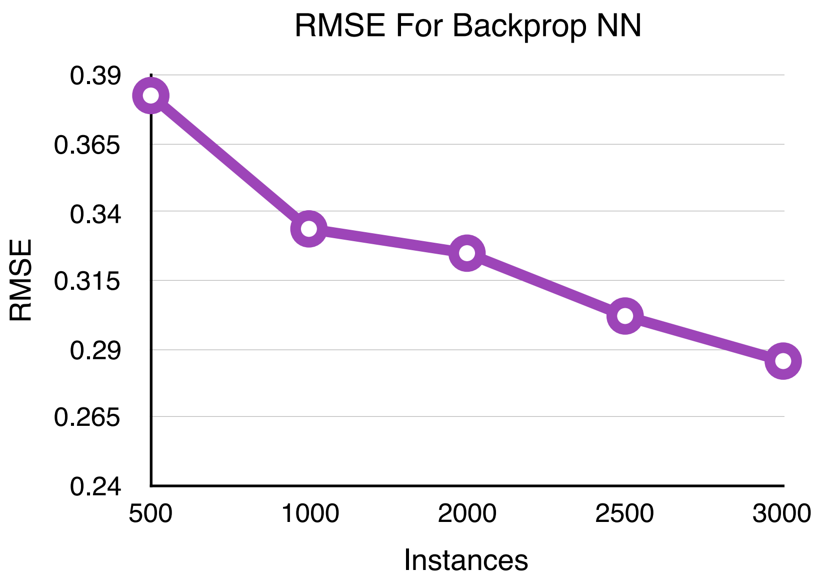




According to the graphs above, it appears that the algorithm that performs the best in terms of reducing RMSE and maximizing the number of correct classifications is GA. It was more consistent than RHC and SA, which is expected according to the performance of RHC and SA later in part two of the analysis. However, there is a trade off for the more accurate results and that is the time that GA takes to run. This could be because the fitness function is evaluated on many members of the population to include multiple variations of children among the parents.

As shown later in this analysis, increasing the number of matings, increases the computation time. The graph directly above is in logarithmic scale to better visualize this drastic jump from RHC and SA to GA.

Also, as the number of iterations increased, the error tended to decrease. This is the expected behavior because more iterations allow more time for the system to manipulate the weights. Although, using the randomized optimization algorithms did not perform as well against the back-propagation neural net from the previous assignment as shown in the graph below.

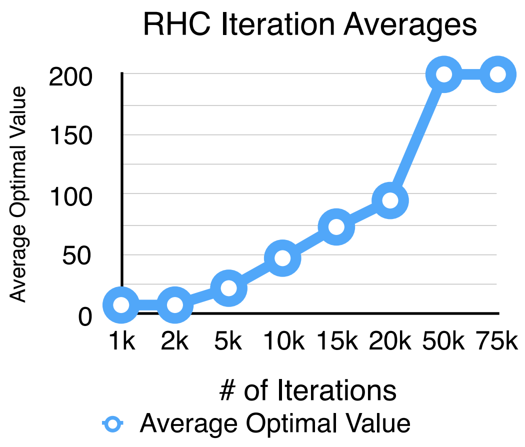


For 3000 instances, it produce a RMSE of around 0.29, which is considerably lower than the other neural network. For each algorithm, the RMSE was greater than 0.7. Perhaps I needed to perform more iterations and given the optimization algorithms more time to arrive to an optimal weight value. In retrospect, I regret not exploring more into the effects of different values for the hidden layers. The value I used in the above calculations was 20. I suspect that I chose a value that was far too large and overfitted the data by over-representing the complexity of the problem so I went back and I tried a lower value of 9 to see if there was a significant impact. Unfortunately, the RMSE was still around 0.7. It did not affect the percentage of correctly classified instances either. It still produced nearly the exact same percentage (at 10000 iterations). By looking at the graph representing the correctly classified instances, I am sure overfitting is a problem because the percentage is beginning to decrease when more iterations are introduced.

<u>Part 2: Optimization Problems</u>

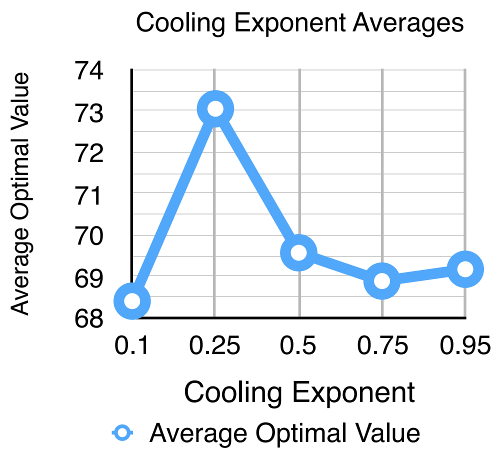
Four Peaks

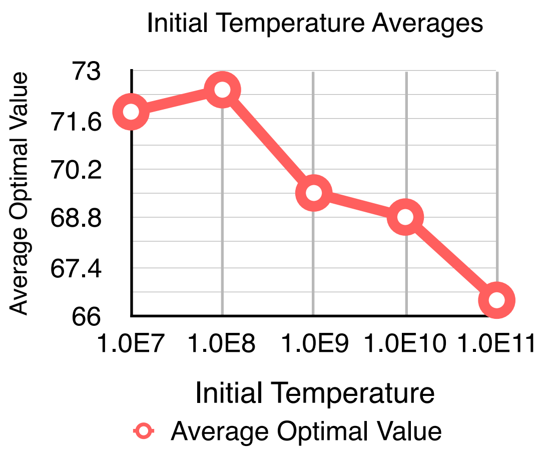
The Four Peaks optimization problem is relatively straightforward in structure. It creates a data set that has four local optima (aka four peaks). I hypothesized the RHC would do the worst, simply because it is very likely to get trapped in local optima, unless we're lucky to pick a starting point somewhere on the true optimum's hill. I tried several different training iterations: 1000, 2000, 5000, 10k, 15k, 20k, 50k, and 75k. The smaller iterations (less than 50k), it failed to find the true optimum of 200. I suspect it behaved as I expected such that it got trapped in one of the local optima. However after 50k iterations it found the true optimum of 200 fairly quickly with a time of around 100 milliseconds or about 0.1 seconds. The above test was using the DiscreteChangeOneNeighbor function. This function simply chooses a random neighbor from the neighborhood of the current point. RHC then continues up the hill if the new point has a greater value than the current point. When I tried the ContinuousAddOneNeighbor function, it failed to find any optima greater than 5 so I decided not to explore into that function any further in the time constraints of this assignment.



The next algorithm I tested was Simulated Annealing since it builds off the weaknesses of RHC. SA illustrates the concept of continuously reheating and cooling a sword makes it stronger. It begins with an initial temperature and gradually cools over time (iterations). When the "sword" (function) is cooled off the final time, hopefully what you are left with is an optimal strength (data point). SA determines to move to a new data point in two cases. The first one is rather simple, it moves to the new data point in the neighborhood if the fitness value of the new point is greater or equal to the fitness value of the current point. The other case is when the new point could be a slight decrease from the current point. Then it decides to move based on the value 'e' raised to the difference of the two points divided by the temperature. The temperature plays a large role in the decision making process of SA. To use this algorithm correctly you need to find a 'goldilocks' initial temperature. If the temperature is too high, then it randomly walks around the data regardless of the fitness value of new point. On the other hand, if the temperature is close to 0, then it simply acts like RHC. I hypothesized that SA will find the true optima in less iterations than RHC because it has the ability to walk over smaller valleys and potentially land on the true optimum. I experimented with different initial temperatures and various cooling exponents. I found that an initial temperature of 1E08 and a cooling exponent of 0.25 performed the best, as illustrated in the graphs.

The other values showed exactly what happens when you choose sub-optimal values for the temperature and cooling exponents. In comparison with RHC, SA converged to an optimum quicker than RHC in terms of number of iterations and real time. It converged to an optimum

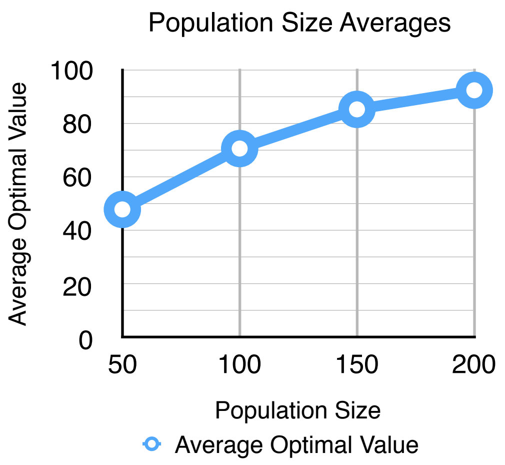


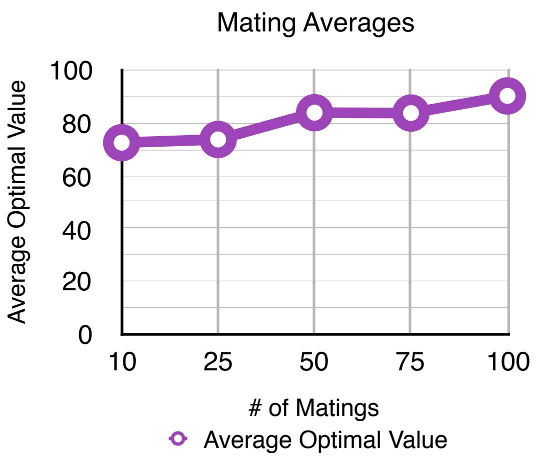


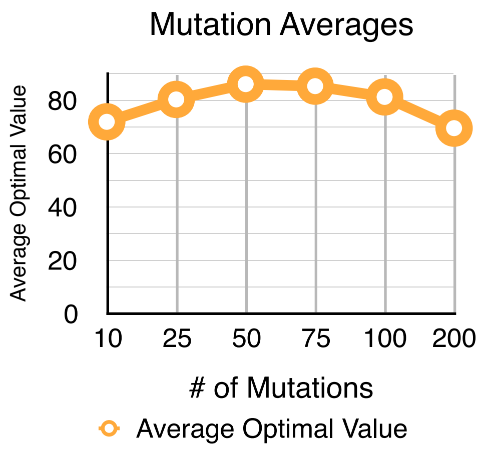
around 43k iterations in less than 0.1 seconds. SA behaved as expected in this type of problem. Where RHC would get trapped, SA could move in and out of those valleys by the liberty of the initial temperature.

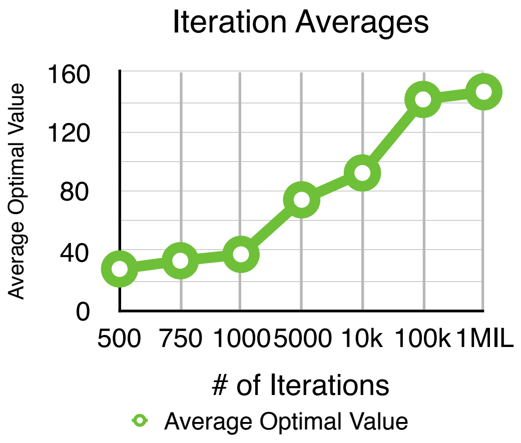
The next algorithm I explored was GA. This algorithm is a bit more involved than the previous two. It follows the analogy of reproduction such that it takes a pair of data points and does some mating process to create a child that hopefully is a combination of the "best" parts of the parents. In terms of the algorithm, it chooses a portion of the data set (population) to "breed" a new population through mating functions. The "best" points in this new population are more likely to be selected for future breeding processes. In terms of the Four Peaks problem, "best" is determined by the value of the point (higher values are better than lower values). In addition to mating functions, mutations could also happen. Mutation is used to maintain genetic diversity among generations of populations. You don't necessarily want just the "best" values in the data set because that could lead to sub-optimal results. Sometimes, as shown in SA, going down a path that seems sub-optimal at first, could lead you to an optimal solution.

The mutation probability can be very helpful in finding an optimal solution, but if it is set too high, then the search algorithm will turn into a purely random search. When I ran multiple tests, this situation began to occur. If the mutations began to approach the size of the population size, the results began to be more sporadic, due to the fact it was turning into a random search algorithm. I found that larger population sizes and a larger mating size contributed to a more accurate result. This makes sense because the bigger the population size, the more accurate of





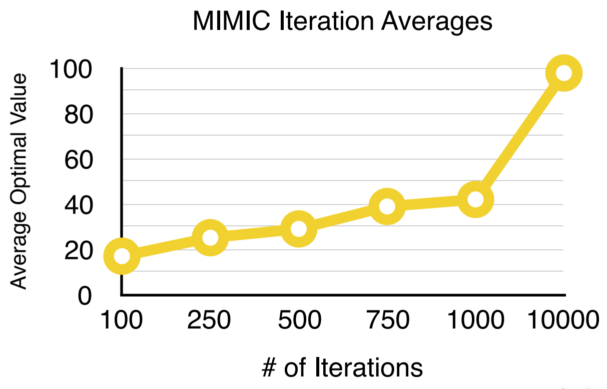


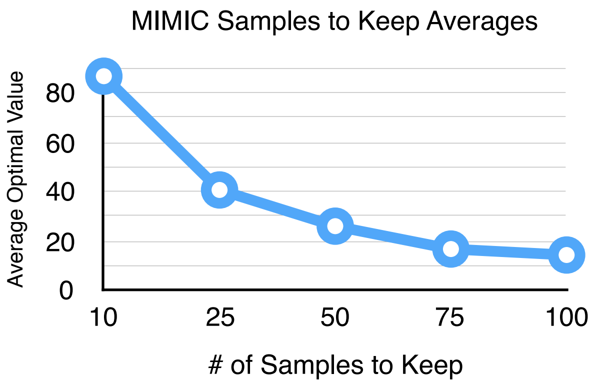


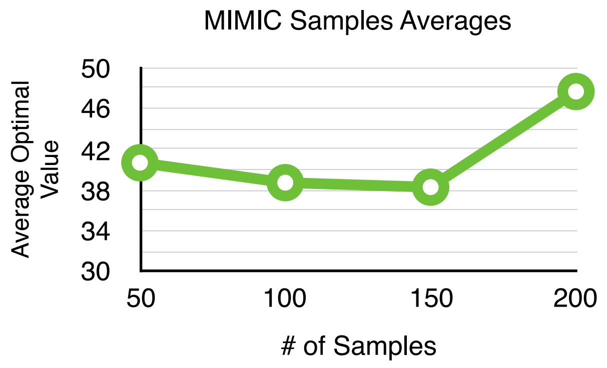
the generalization of the entire data set. Also with more matings, the more likely you'll find "children" that consist of more correct values, thus leading you to a more optimal solution.

In comparison to previous algorithms, GA took a considerably longer time (in real time and iterations) to converge to an optimum. For the Four Peaks problem, GA would not be a good choice due to the time inefficiency. So far SA seems to be the best choice for a problem of this structure, but before any conclusions are made, MIMIC still needs to be analyzed.

The last algorithm is MIMIC, which is fundamentally different from all of the previous algorithms discussed. Instead of focusing on the values of single data points, MIMIC estimates a probability distribution for each sample it takes from the overall data set and takes a proportion of this sample (based on how high the probability is) to continue the iterations. MIMIC shows that is performs very poorly in regard to real time. This is as expected because it needs to maintain the dependency tree after subsequent estimations are performed. However, it did take a smaller number of iterations to converge to the true optimum. I also found that when the samples to keep approaches the size of the sample, MIMIC failed to find the true optimum. This







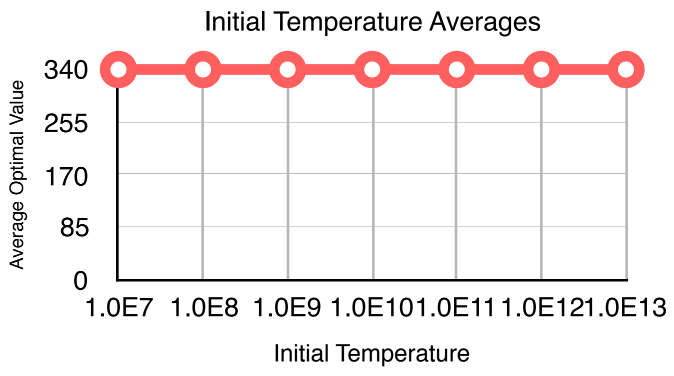
is expected because if you keep almost or all of the sample, then there is no point in estimating a probability distribution for it (takes longer to converge and will probably converge to sub-optimal solution). This behavior becomes apparent in the graphs.

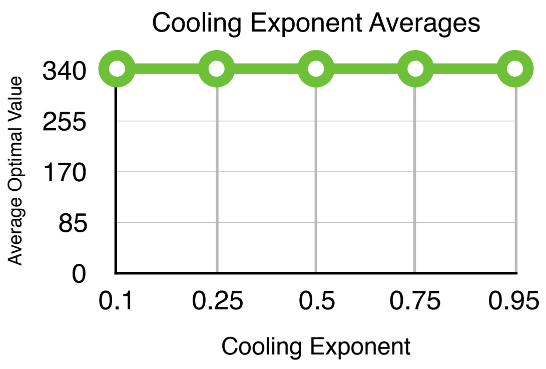
In conclusion, for the Four Peaks problem, SA performed the best due to the fact that it was able to converge to the true optimum in a reasonable amount of time. The other algorithms eventually found the true optimum, but took too long to do it.

Max K-Coloring

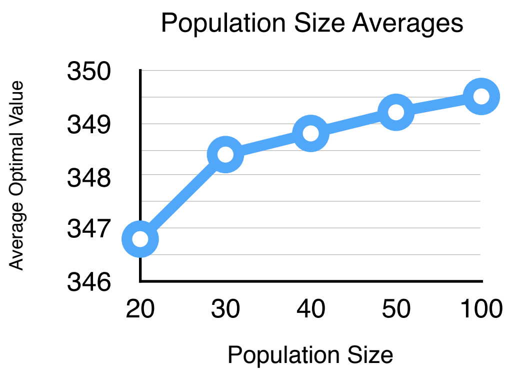
In this problem, the algorithms are to find a maximum k-coloring of a randomly generated graph, where k is some arbitrary integer. Max k-coloring would return the number of iterations need to find the optimal coloring. My hypothesis was that either MIMIC or GA would perform the best, because they have more foresight into the effects of the current decision, MIMIC in the form of dependency trees and GA in the form of choosing the best points from a population that will produce the best future populations.

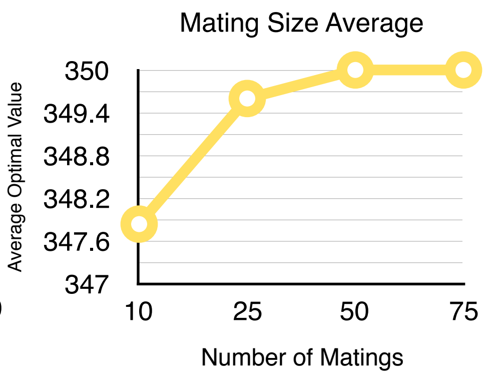
I started with RHC again because I was not confident in it's ability to find the optimal value due to the fact that it only looks at the value of the current point and one of its neighbors at a time. It blindly climbs regardless of what the overall effect of those decisions might be. According to all the tests I ran, my hypothesis was true. Regardless of how many iterations I preformed, it never found the optimal max k-coloring.

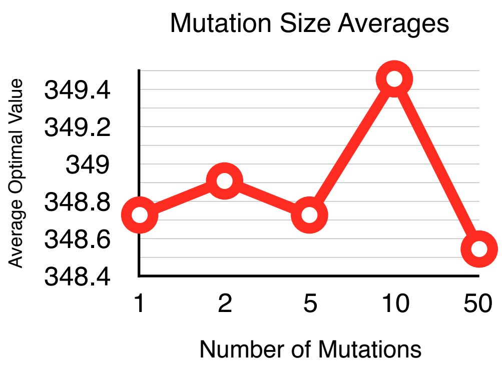


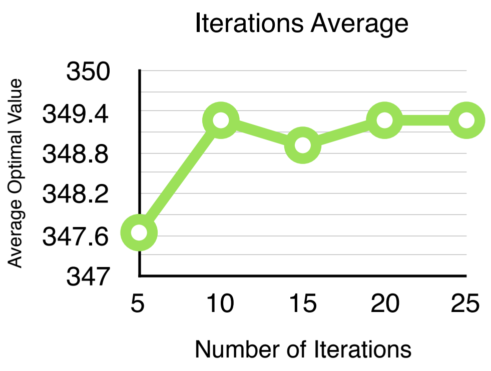


For the same reasons as RHC, SA performed very poorly. It did not find the optimum coloring for the graph regardless of the initial temperature or cooling exponent. I even performed a million iterations and it did not make a difference. It still found an inconsistent coloring assignment. To solve this problem required some foresight into the dependencies of the graph.



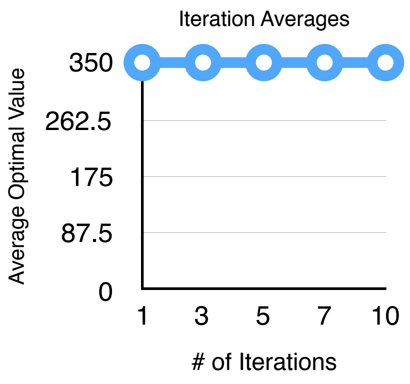


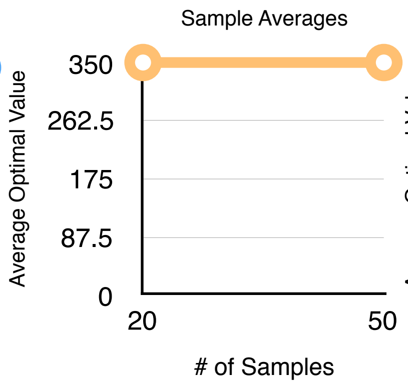


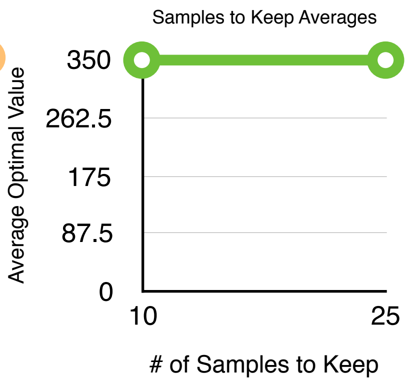


Then I tested GA and I found that it behaved closely to my hypothesis. I found that for greater population sizes paired with mutations around 10 and matings around 50 or 75, it was more likely to find the optimal max k-coloring. When it didn't find the optimal coloring, it was still close to it. I knew this because it would either return 340 iterations or 350 iterations. When it returned 350 then it was the optimal solution. This is shown in the graphs above.

This is an ideal problem for MIMIC and it turns out MIMIC found the optimal max k-coloring for every combination of sample size, samples to keep, and iterations. Also it required a very low number of iterations (1, 3, 5, 7, 10) and an extremely short amount of time to find the optimum. This is as expected because using a dependency tree can quickly determine the effects of the current decision, thus when MIMIC makes a decision on what path to pursue, that choice



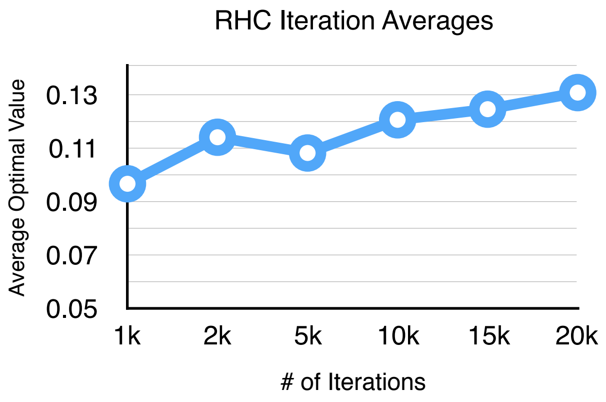




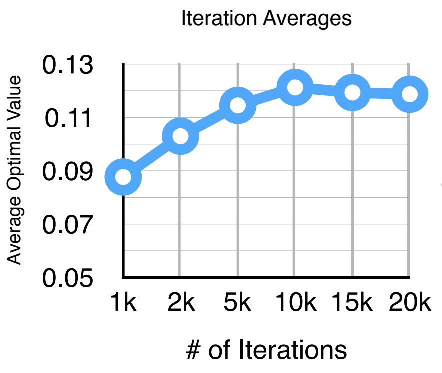
has the highest probability of actually leading to the optimal value. In the graphs above, you can see this exact behavior.

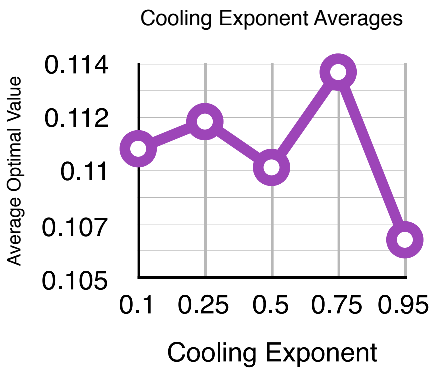
Traveling Salesman

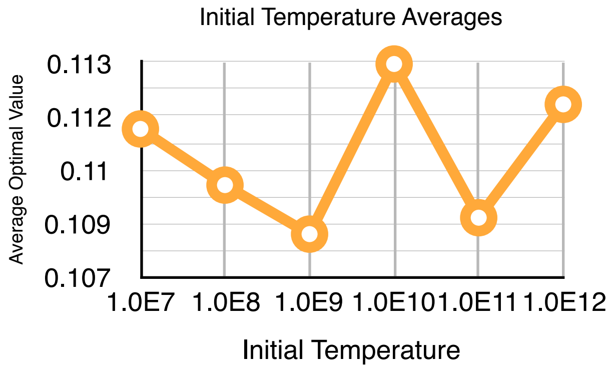
The last problem I used was the Traveling Salesman problem. In this particular implementation, the goal was to maximize the 1/distance the salesman has to travel to visit all the points in the space. Unlike Max K-Coloring, the points in this problem don't necessarily depend heavily on each other, in other words there aren't any constraints on each particular point. Due to this problem structure I theorized that MIMIC might not perform well. I would think that GA would do particularly well because the structure of this problem resembles the type of 2-D space that GA thrives in. As for the other algorithms, I needed to see the test results to make a conclusion.



I began with RHC and began to play with different iteration values. I found that once the iterations increased to very large values, the 1/distance increased as well. This was expected because RHC simply goes from neighbor to neighbor if the new point has a greater value than the current one. This could easily lead to a greater overall distance for the salesman to go, thus leading to a very small 1/distance.

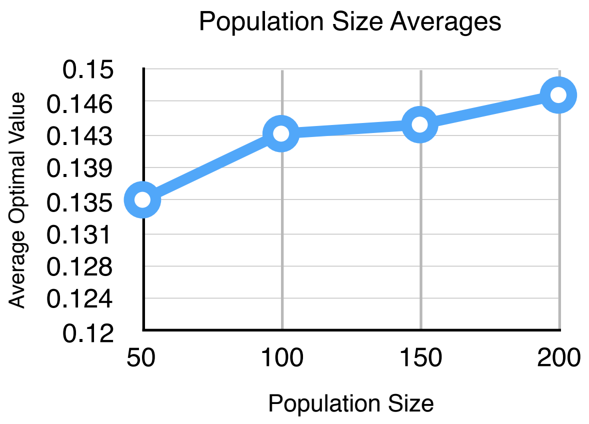


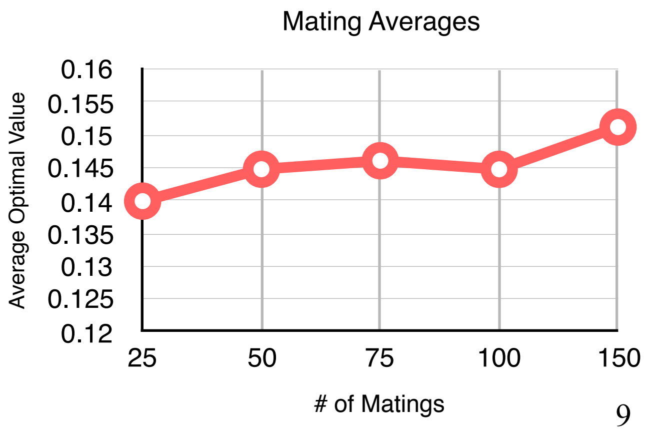


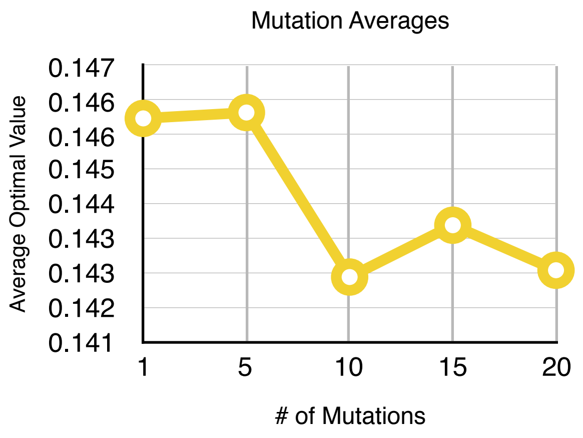


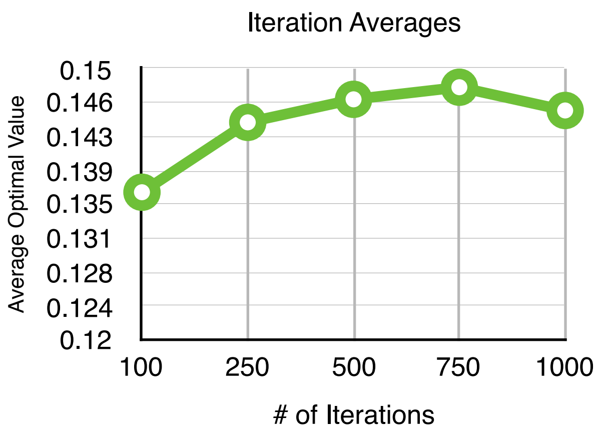
Simulated Annealing fell victim to similar disadvantages as RHC. It found greater distances which contributed to smaller 1/distance values. It produced similar outputs to RHC, regardless of the fact that it should pursue some paths that are not necessarily greater in value than the current point due to the acceptance probability function.

Nonetheless, a typical hill climbing function is not the idea approach to a problem that in reality is trying to minimize distance. Hill climbing seeks the greater fitness value, and in this problem, that proves to be less optimal. SA and RHC required a significant amount of iterations to approach their more optimal results and as you'll see when I get to GA, it was not the most optimal result as well.





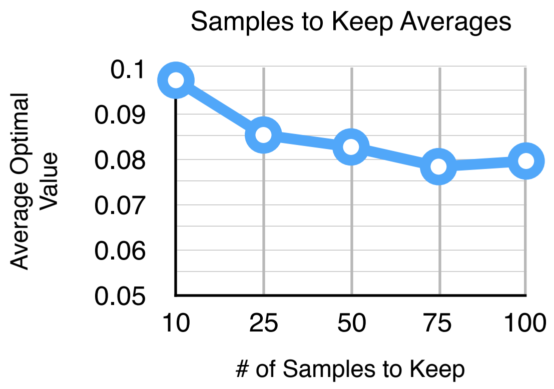


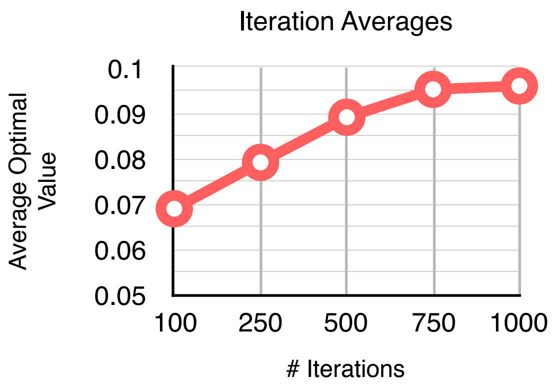


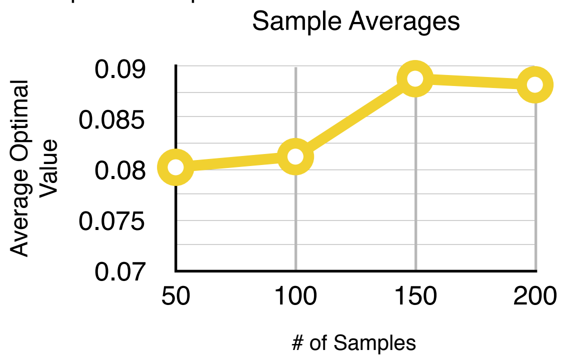
The next algorithm was GA, and this algorithm produced the best optimal values out of the previous algorithms as expected. The mating process in GA proved to be an effective approach to this type of problem. For example, combining a point far along the x-axis with another along the y-axis could potentially produce a more optimal path in a linear direction. GA is good in 2-D spaces such as Traveling Salesman.

As shown in the graphs above, a larger number of matings and population sizes, with a low mutation of 5, resulted in the greatest 1/distance on average. It also converged to the optimum in a relatively short amount of time (in iterations and real time). It only took about 750 iterations compared to the thousands, or even hundreds of thousands, in the previous problems.

Lastly, for MIMIC, it performed as I expected such that it performed the worst out of all the other algorithms in terms of output and time to run. Traveling Salesman takes the form of a sort of scatter plot. It is not differentiable and there appears to be no clear trend to the data as well. Therefore, what is normally the strengths of MIMIC has appeared to work against it. Estimating a distribution for this random data, proved to be wildly in accurate in finding the most optimal path for the salesman.


Samples to Keep Averages
Average Optimal Value
# of Samples to Keep


Iteration Averages
Average Optimal Value
# Iterations


Sample Averages
Average Optimal Value
# of Samples