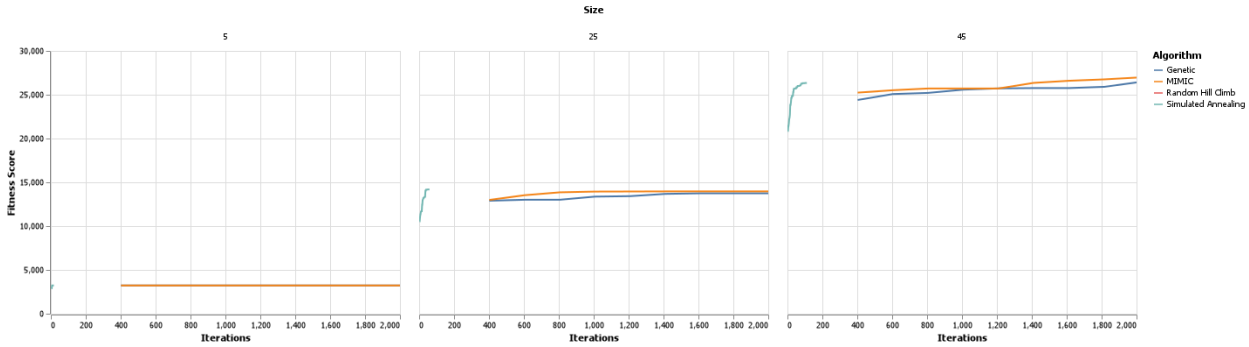
Charles M. B. Fuqua

Randomized Optimization

Randomized Optimization is all about trying to either minimize or maximize a certain equation to find the most optimal case relative to the problem. The algorithms we will examine are called Genetic (GA), Simulated Annealing (SA), Random Hill Climb (RHC) and Mutual-Information-Maximizing Input Clustering (MIMIC). To demonstrate these algorithms we will be trying to maximize 3 different classical optimization problems, Traveling Sales Person, Flipflop and 4-Peaks. Each of these problems was selected with a specific algorithm in mind, Simulated Annealing, MIMIC and Genetic Algorithm respectively. One of the things you have to consider in a given problem is the number of iterations. I didn't pick one specific threshold for my iteration on each problem, instead I picked a dynamic value based on the general vicinity of the algorithm I want to highlight. I chose this because I found that given enough iterations, the GA can beat out most of the other algorithms. This is in hopes of putting everyone on the 'same playing field'. If I was given 1 hour to run as far as I could and you were only given 10 minutes, it is clear who the winner would be, and this is the logic I based my threshold decision on. Once we have highlighted the strengths and weaknesses of these models, we will then use the GA, SA and RHC algorithms to create the weights in our neural network from the previous assignment. This will be in attempts to demonstrate how each algorithm can be used set the weights in a neural network in place of back propagation. We will then compare the results of each of the models to see which one performed the best.

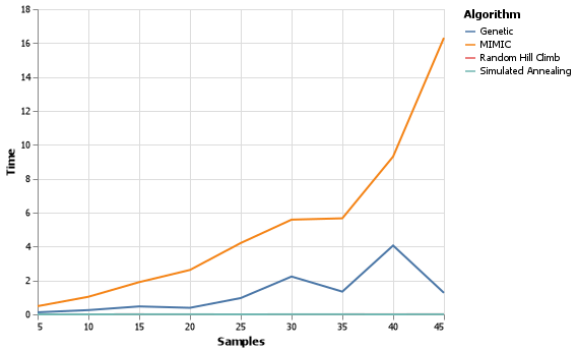
Traveling Sales Person

The Traveling Sales Person problem is all about trying to maximize the shortest route from one city to another, so it is all about getting the cities in the correct order with the shortest distances being used to travel from city to city, in a linked chain. As a reminder, SA stands for Simulated Annealing. I wasn't sure what Annealing meant and I found this definition from the Oxford Languages Dictionary: "heat (metal or glass) and allow it to cool slowly, in order to remove internal stresses and toughen it" which at a chemical level means the molecules are put into straight tightly knit lines. SA is built for creating the short distance in chains. By adjusting the 'temperature' of our model we can heat and cool the dataset to align the datapoints in the straightest shortest lines possible. In order to identify how SA behaves under different circumstances and compared to the other algorithms we will be changing the number of samples or cities (5 -> 50 by 5s), iterations (5 -> 100 by 5s) and max attempts (2 -> 22 by 2s). We won't be examining all of those parameters, but they were fun to explore. As discussed above, we expect this algorithm to shine through on this optimization problem.



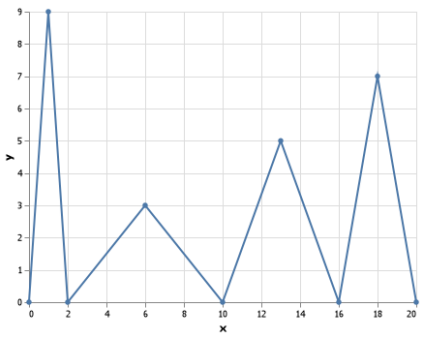
Above is a graph that shows the fitness score per iteration per sample size, although I did cut out some of the samples because the graph was too long to look at. I was a little surprised to see how fast the SA algorithm was able to come to the global optimum. In each of the 3 samples, SA was able to arrive within 200 iterations where the other algorithms took a minimum of 400 before they were able to arrive at roughly the same answer as SA. If I had let this graph go on for as long as it would like, the GA was able to reach a little bit better score, but, in some cases, it

took nearly 10 times as long. It does make sense why MIMIC would show up as a top performer as it also is design to handle and create structure in problems, and as we can see it approaches the fitness score of Sa, just over a larger amount of time. When you are looking for low iterations for a chaining problem, the SA algorithm is the best way to go. With how simple and straight forward this problem is, it is also interesting to look at time. To the right is such a graph. 3 out of the 5 algorithms are indistinguishable from one another, but GA and MIMIC are very clearly defined. Where the GA and MIMIC algorithms have more computations to handle it isn't surprising that they took this long. It is worth noting the exponential growth of MIMIC as the number of samples increase. With such a simple problem and a complex problem algorithm as MIMIC, we see one of the pitfalls of MIMIC shine through which is computation time, when having structure doesn't benefit in the problem. Our GA also had a couple of spikes in computation time, this is controlled by the number of crossovers and mutations we allow the algorithm to perform on the data set, so there is a balance to be struck as we try to output the best fitness score, which might take more complexity which would increase time. These two points of MIMIC and GA apply to the following 2 problems as well.

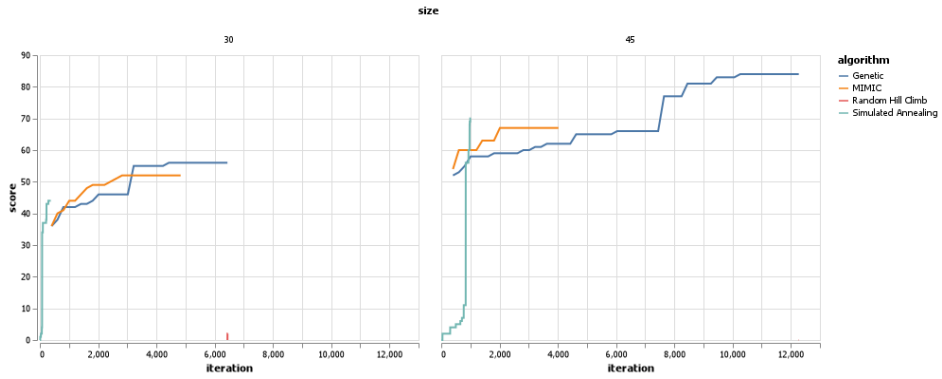


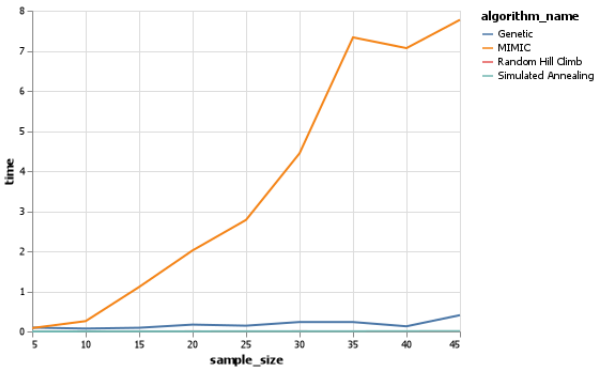
4-Peaks

The 4 peaks problem is a subset of a class of ideas called n-peaks, where n represents the number of peaks in an optimization problem. Consider the graph to the left. Each of these represent optima. Obviously, we can tell what is the global maximum, but to a machine it is hard to find. The basin of attraction for our global maximum



is only 10% of the of our space, thus meaning our algorithms have a 90% chance of getting stuck in a local optimum. SA, RHC, and MIMIC would all struggle in this type of scenario as the first two tend to focus on randomly searching or exploring the space and MIMIC as the next generated subset of data may not include this small 10% of the data. The GA though should be the best one. Think of the algorithm like this, we have a 3D topographical map and we are going to try to find the highest peak. We make a plane starting at sea level and slowly raise that plane until we reach the last peak that is touching the plane, we then follow it to the top and find the maximum. Now, the math that goes into this is much more complex dealing with mutations (a change within a local maximum) and cross-overs (a combination of the best values from different local maxima), but you get the idea. Here is a graph that demonstrates the score per iteration over a two different sample sizes. It is very clear how quickly the GA can learn and perform better than the other algorithms. It isn't surprising to see MIMIC follow GA as the ideas and methods found within the MIMIC algorithm can be directly mapped to ideas of cross over, mutation, and other aspects found within GA. To the left is the time graph. As the nature of this problem is not as complex as the Traveling Sales Person, we wouldn't expect to see the algorithms take as long. We see that in the axis and in the MIMIC algorithm, although it is interesting to note that the increase in time appears to be more

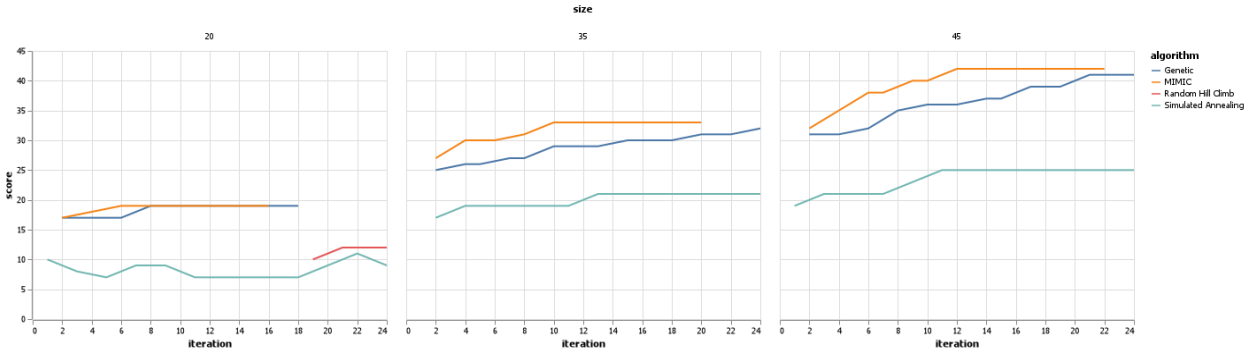




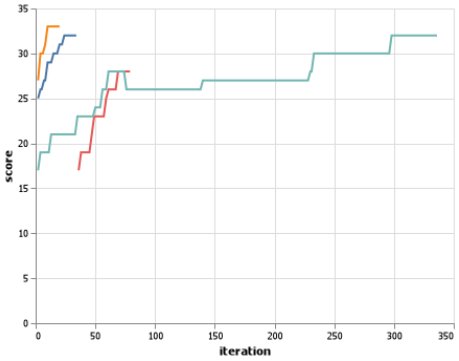
linear than exponential in when compared to the previous problem. This is most likely due to the complexity of the scenario. Again, remember the complexity balance between GA as well, although we see in such a simple problem, a complex model is not needed and we are able to still get a good score without having to sacrifice computation time. This helps us to know that there are clear peaks within the data and few crossovers and mutations are needed to reach the optimal value, which is the nature of this problem.

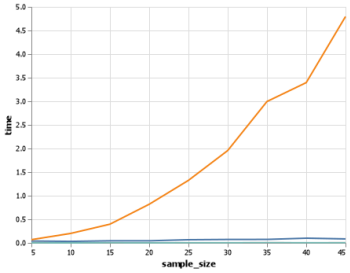
Flip-Flop

The Flip-Flop optimization problem is all about the alternation from one number to another in each set {0,1}. This sounds like a problem that would be best suited for SA as it deals with chaining, but one flaw is presented when we look at this problem. The SA algorithm tries to find the best possible value in a distinct list, including trying to find the best between 1010101 and 0101010. Both are optimized based on the conditions of the problem, but SA has a high chance of struggling to come to a decision when these values are placed in front of it as its bias is to select one best value. MIMIC, on the other hand, deals more with structure than with sequence. MIMIC asks the question “Are these meeting the criteria of the problem and if so, we can select all of the values that fit this problem the best” whereas SA asks “is it better to have 101 or 010 because there can only be one”. MIMIC follows more closely what our logical reasoning would be as humans.



Above is the resulting graph from this problem. Again we see GA and MIMIC sticking close to each other, but the strength of MIMIC shines through on this problem as in all of the other problems MIMIC was just below our GA for that problem. One item that was interesting for me was RHC. I know it wasn't one of the ones we were supposed to highlight, but this algorithm barely makes it into the first graph but isn't present in the other two. If we zoom out on the middle graph we have the graph to the right. This algorithm wasn't able to produce a fitness score until both MIMIC and GA had stopped completely. This also gives us a good perspective on how well MIMIC was able to do when compared to SA, the algorithm which theoretically should do just as well as MIMIC. We see SA falling into the trap that we had mentioned before, it can get stuck evaluating 101 and 010 to see which is better. When you are looking for structure and don't care as to how it looks, MIMIC is the way to go. Alas, MIMIC is unable to get away from its biggest pitfall, computation time. Again, we see what looks to be the start of an exponential growth and can only lead to intense computation times as the number of samples grow infinitely large. Again, in such a simple problem of flip flopping numbers, a complex GA is not needed and so we can have a low computation time.

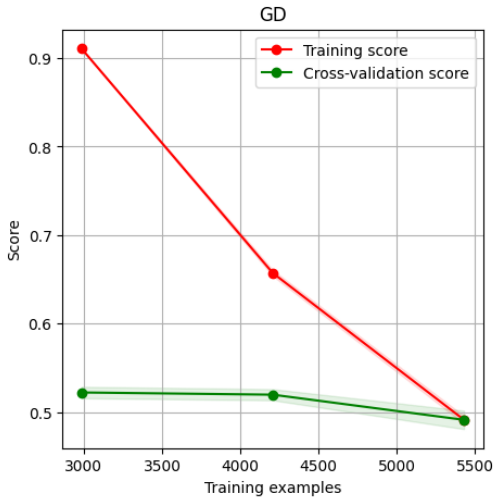




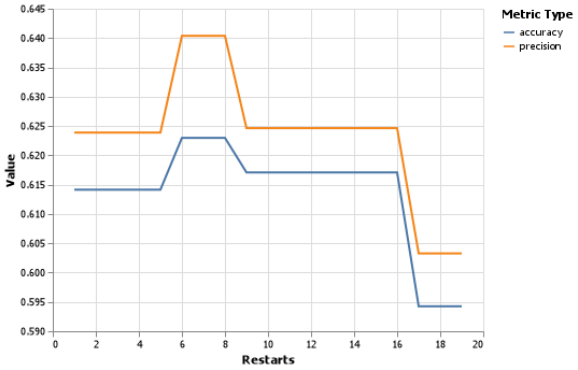
Perfecting The ANN

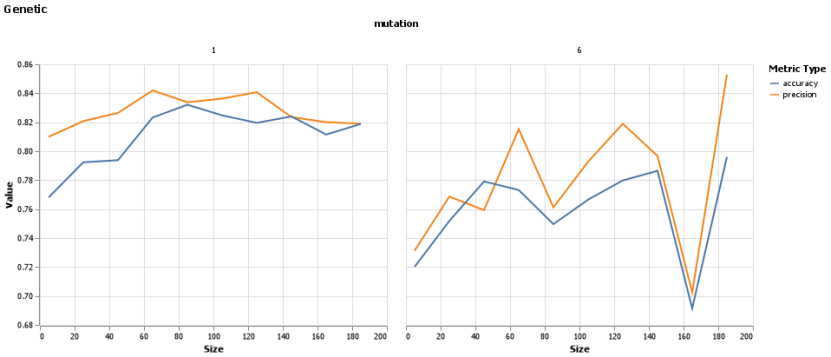
Now that we have explored these algorithms, we can put them into practice by trying to perfect the weights of an ANN. Now, I couldn't use my exact ANN from the previous

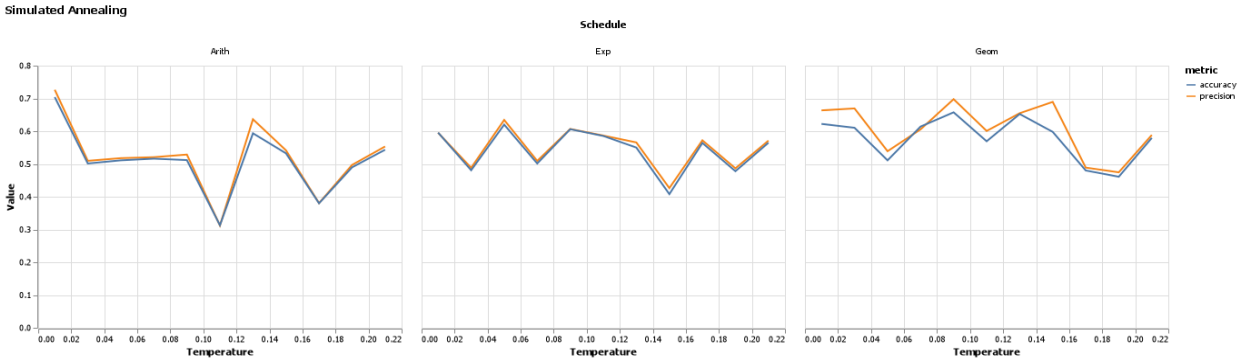
assignment because one of the activation functions I had used from Sklearn wasn't available in mlrose-hiive, the logistic activation function. So, with that in mind here are the new base values to try and beat; Time: .15 Precision: .53 Accuracy: .52 and a learning curve. Given what we have seen, and the nature of ANNs I would expect the GA to perform the best.



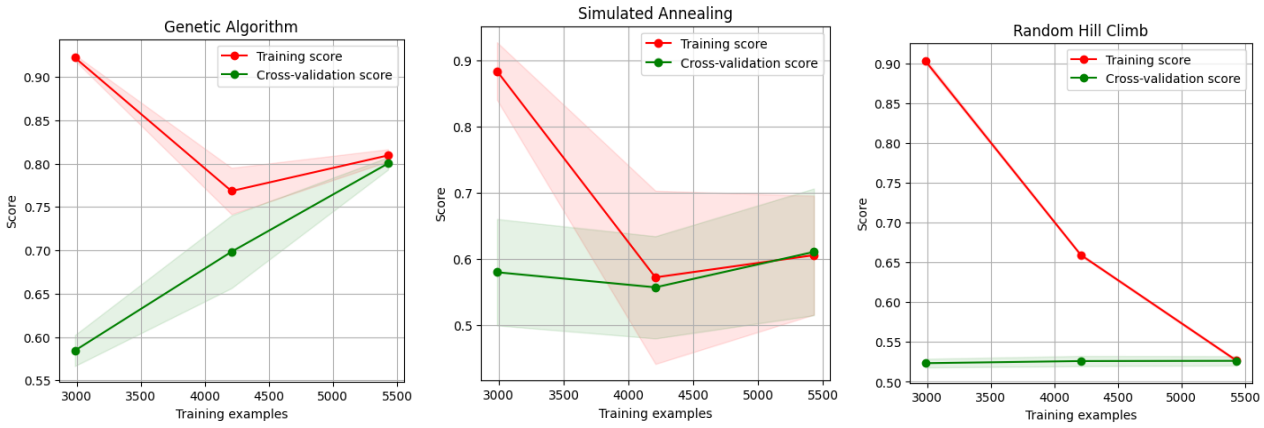
Each of the algorithms have some different values associated with them, so we will first attempt to find the best values for each algorithm before I get into the learning curves. Here are 3 graphs that show the scores over different parameters. I did this to give these algorithms the best possible chance. Based on these graphs we will be using 6 restarts for RHC, a schedule of Geometric Decay with a temperature of .09 for SA and 1/100 or .01 mutation_prob with a population size of about 120 for our GA.







Below are the resulting learning curves from the algorithms, remember the learning curve for our base case is presented above. The times for training are as follows: SA = 2.078 RHC = 11.05 & GA = 23.408. The training times seem consistent with the values we were seeing up above, except for RHC being unusually high, but as noted below, the algorithm didn't do very well so it most likely got caught in a local minima, and a very poor one at that. As for the learning curves below, due to the convergence of the training score and cross-validation score we can trust these values, and even we might use them in our actual algorithm. It isn't surprising to note that the genetic algorithm did well, when given as many iterations as needed the algorithm was able to perform well in and of itself in each of the optimization problems. Seeing as how a neural network is all about combining data into nodes, mutating them then allowing the information to cross over in a dense network, it isn't surprising to see that our GA performed better than the rest. In a high dimensional dataset like this one, it is easy to understand why RHC did so poorly. With higher dimensions there are bound to be many pitfalls and local optima to get trapped in, with only a small number of iterations and restarts allowed, we wouldn't expect it to perform well. If we were to only work in a small dimension for a simple linear model, I would believe RHC would have a change.



Overall, we have learned some interesting things about each of these 4 algorithms. We learned that RHC is rather simple, checking its neighbors to see which value is better then 'climbing' that hill until it reaches the top and if it is allotted a randomized restart it goes through that whole process again. Due to this, the algorithm suffers in complex spaces as sometimes you have to go down a mountain to get to the highest peak. We learned that SA is amazing with problems that require it to put values in order based on some metric, but only as long as there is one optimal answer. We found that in the case of flip-flop SA suffers because there are 2 global optima that are exactly the same 101 and 010. This brings us to MIMIC which provides structure to any problem and attempts to share information learned in one iteration to the next allowing us to remember where we have been, so we don't have to spend time searching over those values again. Sometimes it is important to form densely connected graphs that show how each point interacts with the others giving us their mutual information or what is called a maximum spanning tree. Lastly, GA performed well on all these problems due to its ability to combine attributes and create new planes for us to work in, but at what cost? Time, Complexity and Data. Due to how different variables are interrelated (crossover) and how some of them can even have mutations performed to make them better, this makes the complexity rise and time right along with it. Each of these algorithms have strengths and weaknesses, knowing which one to use comes down to the nature of the problem, and in our case Traveling SalesPerson was SA, 4 Peaks was GA and flip flop was MIMIC (sorry RHC, but you are a great foundation for understanding how these other problems work). As always, the most important step in machine learning is being able to understand your problem before you pick the solution.

Works Cited

AuthorLastName, FirstName. *Title of the Book Being Referenced*. City Name: Name of Publisher, Year. Type of Medium (e.g. Print).

LastName, First, Middle. "Article Title." *Journal Title* (Year): Pages From - To. Print.