Monira Khan
CS 4641: Machine Learning
March 1st, 2019

Randomized Optimization

The purpose is this assignment is to assess randomized optimization algorithms on a classification dataset–those algorithms bring randomized hill climbing (RHC), simulated annealing (SA), a genetic algorithm (GA), and MIMIC. For the last assignment, I was using sklearn library in order to implement the classification problems, but sklean does not provide the required optimization problems for this assignment. So, I used the ABAGAIL library in order to implement all the algorithms on the chess game dataset from Assignment 1. The chess game dataset consists of board-descriptions of a chess game between a team of a king and rook and a team of a king and pawn. The instance is classified as either win when the white king and rook team win and no win otherwise. The dataset is a binary classification problem. I had already converted the dataset into integers from the last assignment, so the ABAGAIL code worked without much more code on my part. Below, you can see the graph displaying the data distribution of the chess game dataset.

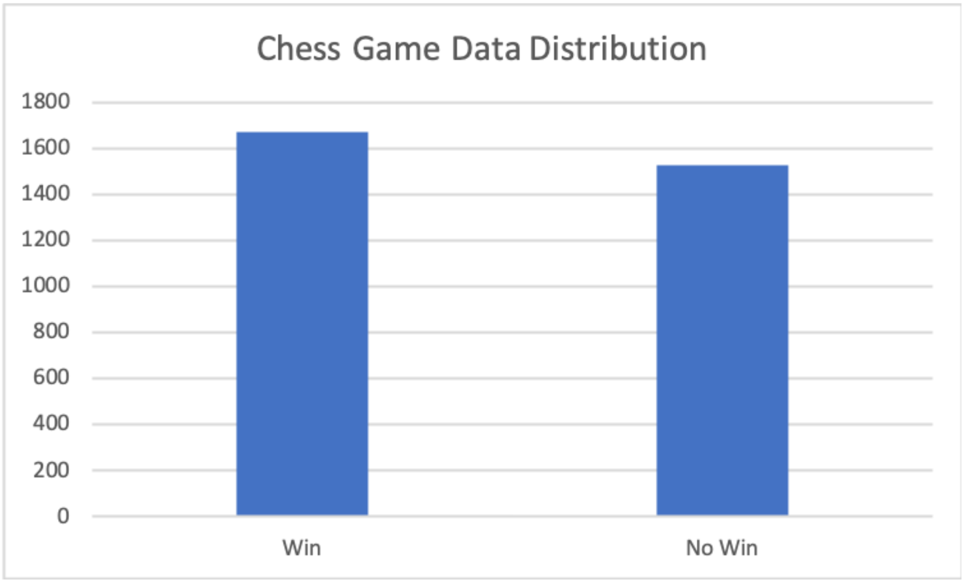

Figure 1: Data Distribution

Optimizing Weight for Neural Networks:

In practice, the backpropagation algorithm is the most normally used learning technique for neural networks' weight optimization. But for the purposes of this assignment, I replaced backpropagation with randomized hill climbing, simulated annealing, and a genetic algorithm in order to optimize weights of a neural network training on the chess dataset. The table below reports the accuracies and overall performance of each of the optimization problems with 1000 iterations.

	RHC	**SA**
		Temperature=1e11,
		Cooling=.95
			GA
			Population number=200,
			Crossover number=100,
			Mutation number=100

Training Accuracy	68.65%	51.272%	63.523%
Training Time	36.719s	36.090s	271.979s
Testing Accuracy	81.939%	56.998%	73.104%
Testing Time	0.009s	0.005s	0.005s

From the overall standpoint, RHC and GA performed very well on the dataset, while SA did not. It's very interesting that SA did not even improve very much from the training accuracy to the testing accuracy while RHC made a significant jump. It is also interesting to note that all the testing accuracies are better than the training accuracies. To get a better look and each of the optimization's performances, I graphed their learning curves below–all trained over 1000 iterations:

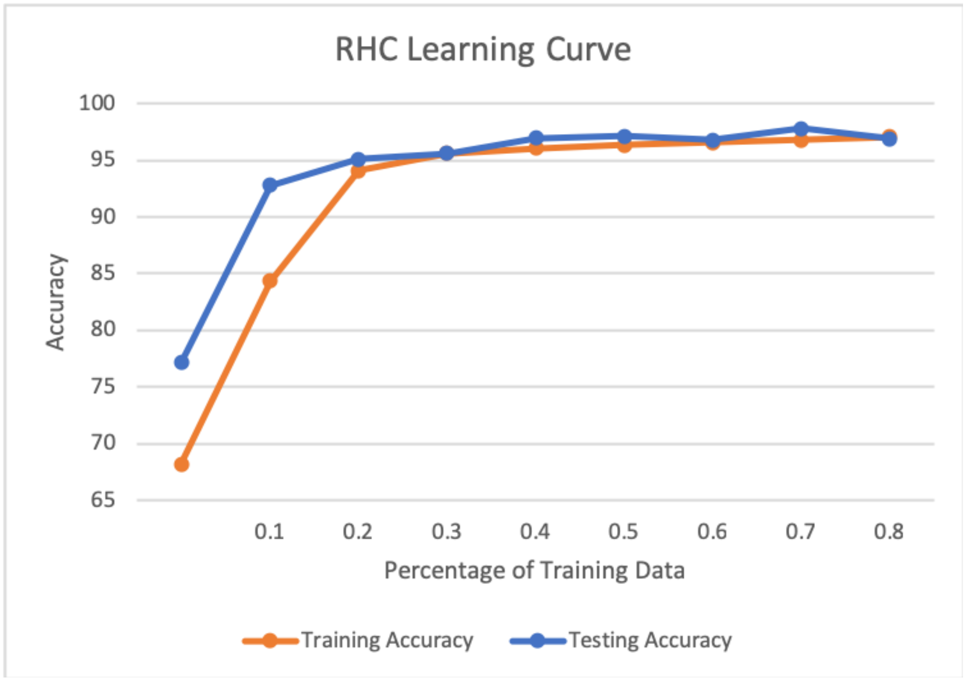

Figure 2: Randomized Hill Climbing learning curve

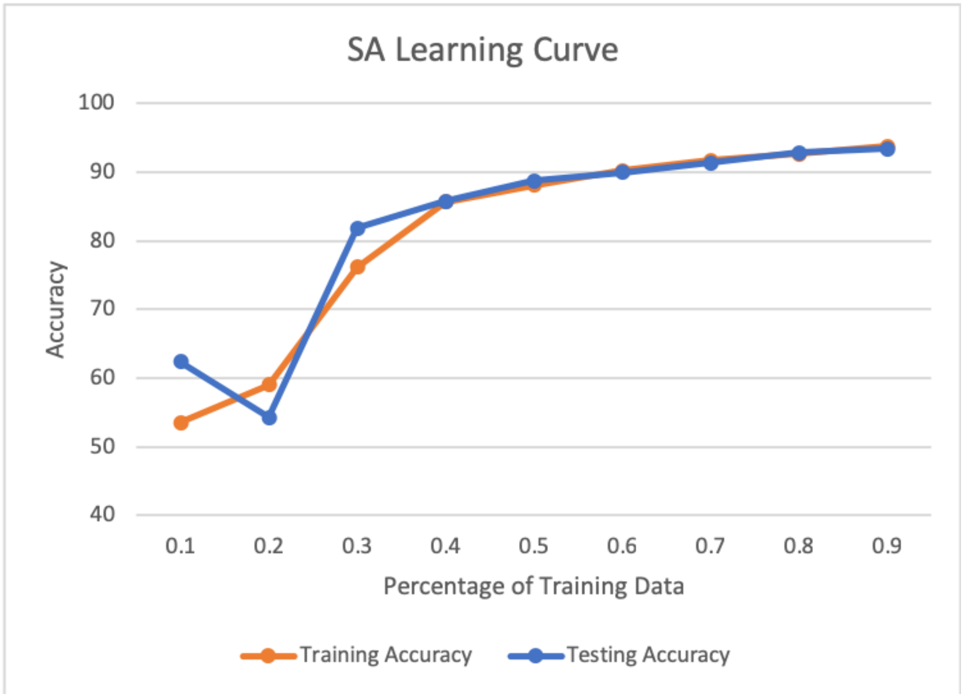

Figure 3: Simulated Annealing learning curve

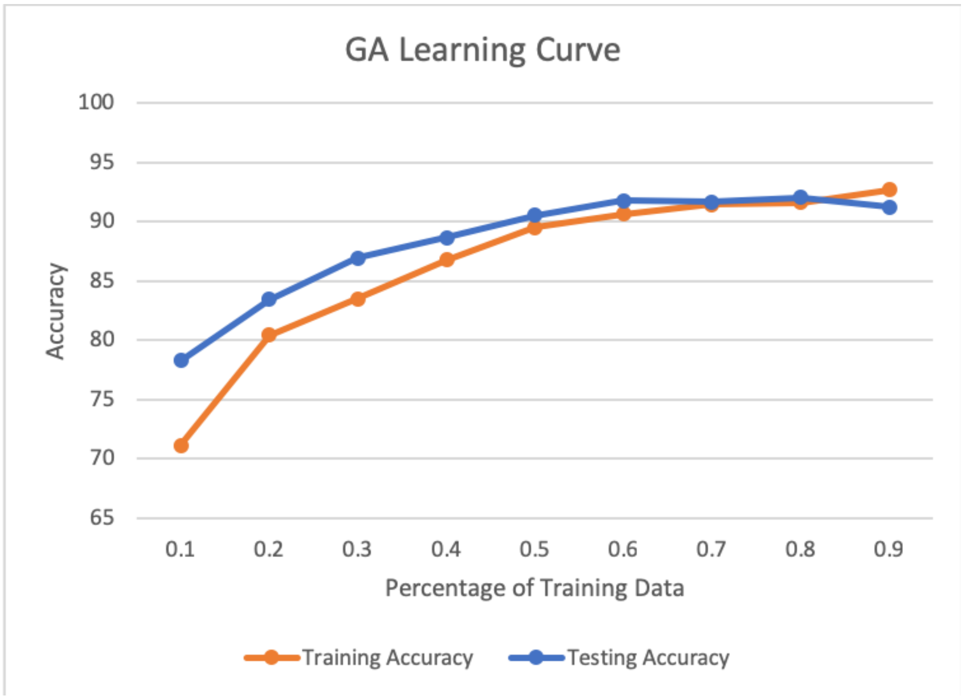

Figure 4: Genetic Algorithm learning curve

Interestingly enough, the genetic algorithms learning curve is the worst performing from all three of the optimization problems. This is counterintuitive to think because one would assume the crossover function utilized by GA would give it an advantage over the other optimization algorithms. The cross over function combines two parent instances to create children instances that would hopefully perform better or as well as the parents allows–by the evolutionary theory of natural selection–the most fit parameters to propagate into the next generation. The crossover function in question is the default crossover function ABAGAIL performs. An interesting question would how over crossover function would perform–like the two point and k crossover. But in addition to GA performing the worst, the GA is the most time-consuming algorithm, and would not be fit for situations in which time is an important factor.

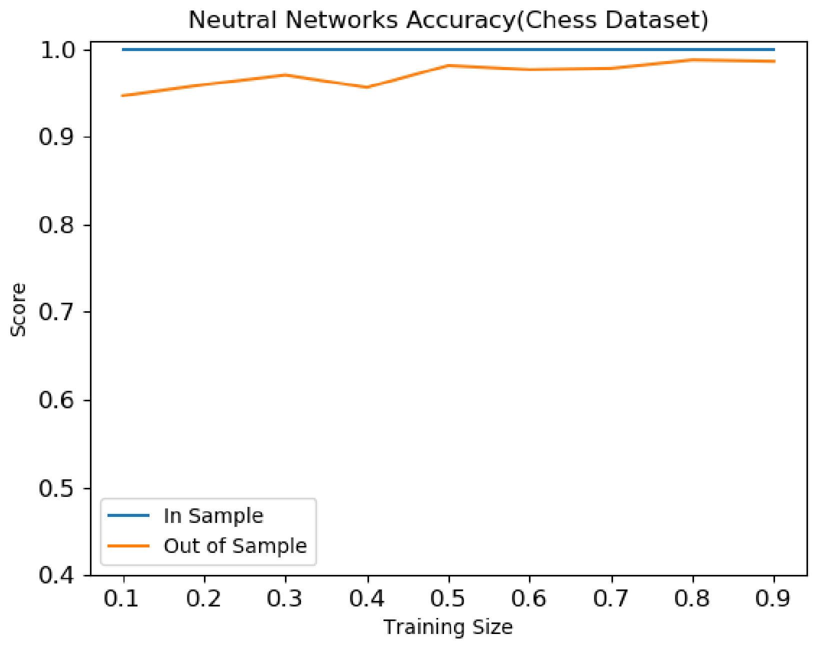

Figure 5: NN Performance on Chess Dataset (From Assignment 1)

Another interesting note, randomized hill climbing and simulated annealing learning curves got the closest to the learning curve of the normal neural network's performance on the chess dataset from Assignment 1. They get close with about 50% of the training data. For both randomized hill climbing and simulated annealing, it seems the algorithms eventually converge to an accuracy with more and more training data. This could signify that the dataset has low variance. It is also interesting to note that randomized hill climbing could easily fall prey to local maxima, but with 1000 iterations, it seems to be enough to explore the dataset to find the true global maxima. Similarly with simulated annealing, 1000 iterations allows the algorithm to have a high probability of finding a fit solution.

Traveling Salesmen Problem:
For the first experiment I decided to run with the classic traveling salesmen problem with all four algorithms–including MIMIC. The traveling salesmen problem is a problem of finding the optimal–as in the shortest–distance between a variable number of cities starting at an origin city and ending by returning to the origin city. A summary of the optimization algorithms' performances is shown in the table below for 100 cities.

	RHC	**SA** Temperature=1e11, Cooling=.95	**GA** Population number=200, Crossover number=100, Mutation number=100	**MIMIC** Samples=200, Samples to keep=10
Inverse Distance	0.0763	0.0692	0.1041	0.0524
Time	0.183s	0.22s	9.1s	279.4s

From the table above, it can be seen that the traveling salesmen problem is optimized by the strengths of the genetic algorithms. In order to delve into the performances of all the optimization algorithms performances for the problem, I graphed them against number of cities.

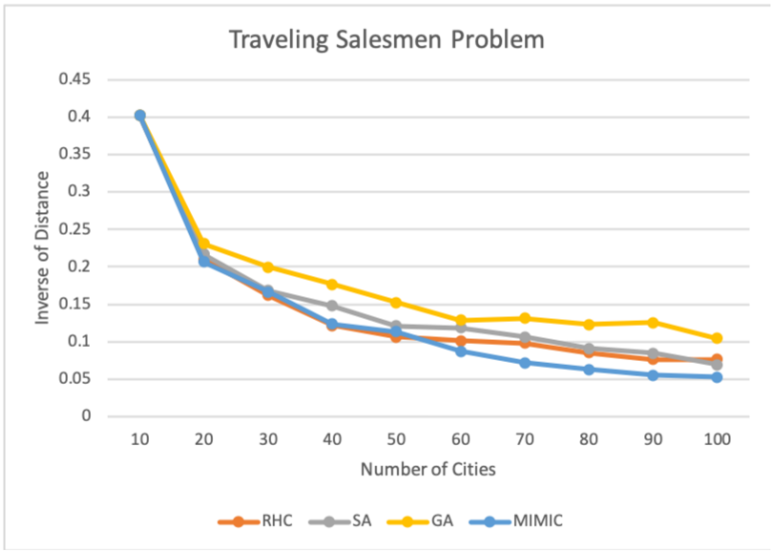

Figure 6: Traveling Salesmen Problem

As can be seen from the graph above, GA outperforms all the other algorithms pretty consistently over all the number of cities. As the number of cities decrease, all the algorithms perform better, but as the problem get harder and harder, GA continues to perform better than the rest–MIMIC performing the worst and taking the most time. It's very interesting how this problem plays on the strengths of GA, because GA is able to pick apart the problem and optimize the parts and combining them. So, GA can optimize the paths between certain cities and then combine all the optimized paths for an overall optimized path hitting all the cities. On the other hand, MIMIC tried to optimize the whole path at once by considering the structure of the entire problem, which costs time and, obviously, accuracy.

Continuous Peaks Problem:

The second problem I chose was the continuous peaks problem. This problem is set in a 1D space with a number of bits, or peaks. The optimization algorithms search the space for the best local optima it can find in the space. The optimal value in this problem is when there are greater than T continuous bits set to 0 and when there are greater than T *continuous* bits set to 1. All the algorithms performed fairly fair timewise on this problem, because it is so easy to converge on an optimum–since there is a number of then–, so for this problem, I will skip the table with the runtime. Below, I graphed all the algorithms performance as a function to the number of peaks.

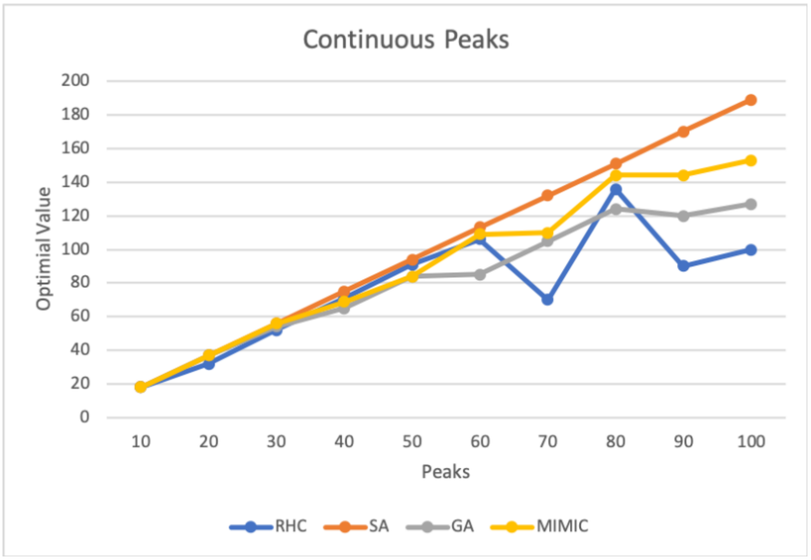

Figure 7: Continuous Peaks Problem

For this problem, it is obvious to see that simulated annealing performed outstandingly, consistently as the number of optima increase. While the rest of the algorithms converged on some local optima, SA was always able to find a better optimum and in reasonable, quick time. This problem beautifully highlights SA's strengths because it is engineered to search large spaces and converging on a global maximum rather than local maximas. SA begins the random search of the large space, taking advantage of slowly cooling down in temperature and only moving to places with *higher* fitness by bouncing between exploration and exploitation! By decreasing slowly, SA has the chance to explore the temperature before cooling.

Four Peaks Problem:

The third experiment I decided to run is the four peaks experiment. This is different from the continuous peaks problem in that we are given an N-dimensional input vector $\vec{X}$, the four peaks evaluation function is:

$$f(\vec{X},T) = \max\left[tail(0,\vec{X}), head(1,\vec{X})\right] + R(\vec{X},T)$$

where

$$\begin{aligned} tail(b,\vec{X}) &= \text{number of trailing } b\text{'s in } \vec{X} \\ head(b,\vec{X}) &= \text{number of leading } b\text{'s in } \vec{X} \end{aligned}$$

$$R(\vec{X},T) = \begin{cases} N & \text{if } tail(0,\vec{X}) > T \text{ and } head(1,\vec{X}) > T \\ 0 & \text{otherwise} \end{cases}$$

The function presented above has two global maxima which are achieved by T+1 leading 1's followed by all 0's or T+1 leading 0's followed by all 1's. There are also local maxima available for the algorithms to fall prey to–a disadvantage to SA. The following table summarized each of the algorithm's performances after 3000 iterations.

	RHC	**SA** Temperature=1e11, Cooling=.95	**GA** Population number=200, Crossover number=100, Mutation number=100	**MIMIC** Samples=200, Samples to keep=10
Optimized Value	19	21	45	141
Time	0.001s	0.001s	0.382s	57.142s

From the table above, it is obvious to see that this experiment really capitalized on MIMIC. In order to get a better look, I graphed each of the optimization algorithm's performances on the problem over iterations in the following graph.

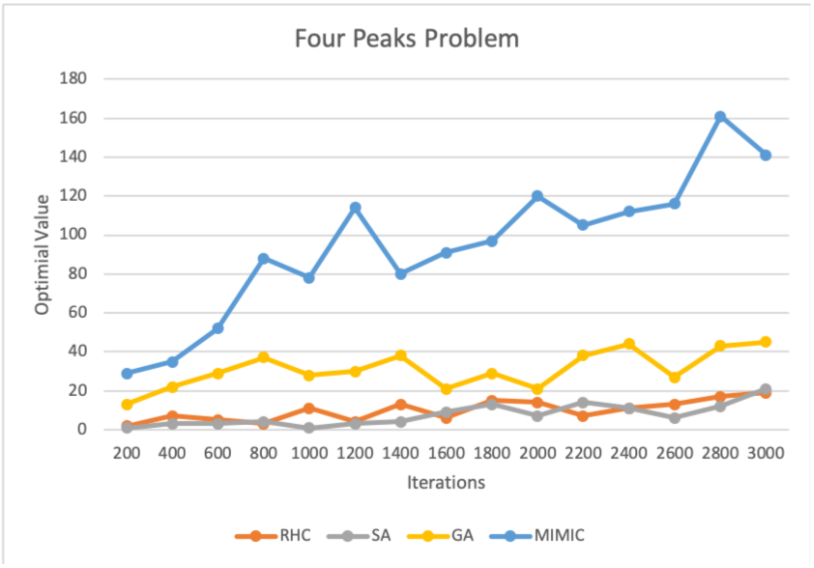

Figure 8: Four Peaks Problem

The MIMIC algorithm out-performed all the others consistently throughout all the iterations, with GA coming in second and RHC and SA being undiscernible for which algorithm is third or fourth. Four Peaks problem is meant for an algorithm that utilizes the whole structure, making MIMIC the optimal algorithm for this problem. MIMIC, using its consideration of structure, is able to evade local optimum and convey points that are better than other points and, in the end, the global points or points! This problem shows very well how the other optimization algorithms are getting confused with local optimum and with the fact that this problem has *two* global optima. For the other optimization algorithms, it is hard to discern which point is go with since they get inputted one point and just output another point, not encapsulating the whole landscape of the problem. But, the great performance of MIMIC, unfortunately, costs major time. As can be seen form the table, all the other optimization problem run in almost no time, while MIMIC takes almost a whole minute. There is always a price with great power. It takes time to consider *the* *whole structure* of the problem at each iteration, while consider one point and surrounding points takes almost no time.