Assignment 2

A few notes:

- All of the below experiments were conducted using **mlrose** library.
- Except for the section that specifically studies the correlation between fitness score and Randomized Hill Climbing allowed restart number, Randomized Hill Climbing methods below all use a restart number of 10

Part 1: Three Optimization Problems applied with Randomized Hill Climbing, Simulated Annealing, Genetic Algorithm, and MIMIC

Optimization Problem 1: Four Peak

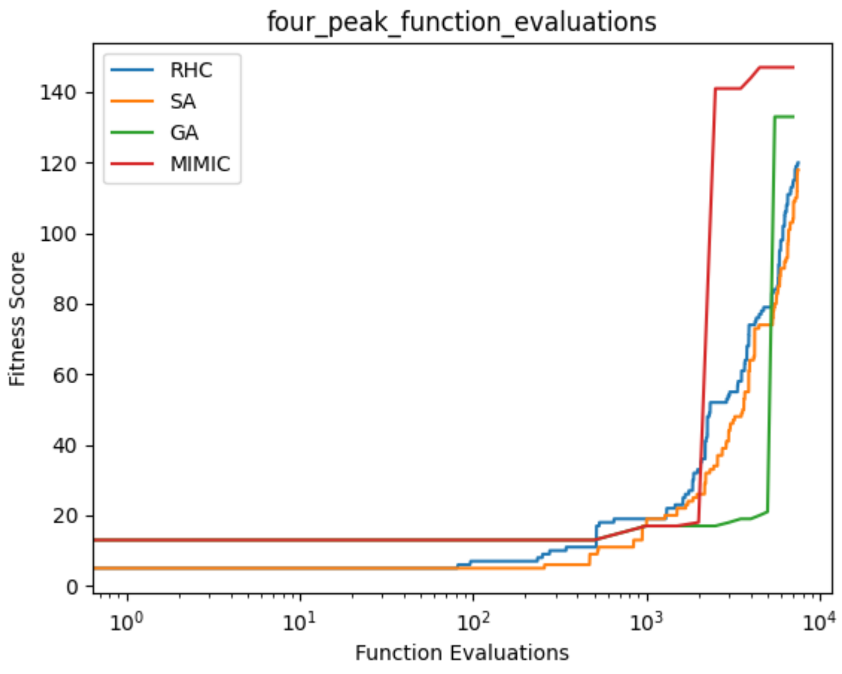
I would like to use Four Peak to highlight the optimization technique of **MIMIC**. Fitness for Four Peak is defined as: given n-dimensional state vector x where each dimension could be 0 or 1, and given parameter T, then Fitness(x, T) = max(tail(0, x), head(1, x)) + R(x, T) where tail(b, x) is the number of trailing b's in x, head(b, x) is the number of leading b's in x, and R(x, T) = n, if tail(0, x) > T and head(1, x) > T, otherwise R(x, T) = 0.

In Four Peak, I specifically limited the number of function evaluations that each of the four algorithms can use to 10^4. A function evaluation is defined as assessing a candidate's fitness. In the real world, there are some "function evaluations" that do take a long time to perform, for example, evaluating a rocket launch simulation's fitness. Hence, there's need to study the performance of various optimization algorithms given limited number of function evaluations allowed.

For Randomized Hill Climbing and Simulated Annealing, one iteration equates to one function evaluation because for each iteration, these two algorithms only pick one neighbor and subsequently assesses its fitness. On the other hand, given a population size of 100, for each iteration, Genetic Algorithm and MIMIC each conduct 100 function evaluations (1 function evaluation for each candidate in the population of 100). Therefore, given the 10^4 function evaluation limit and a population size of 100, Genetic Algorithm and MIMIC each is allowed to perform 100 iterations only (10^4 divides 100).

The Four Peak problem that I designed has N equal 120 and T equal 0.1. I purposely chose a relatively large N compared to the study done in [1], because the basin of attractions for the two local maxima (all 1's and all 0's) become increasingly larger as N increases. It would be interesting to assess each of the four randomized optimization algorithm's ability in escaping the two basin of attractions for local maxima.

Below graph demonstrates each of the four randomized optimization algorithm's performance given limited function evaluations of 10^4:



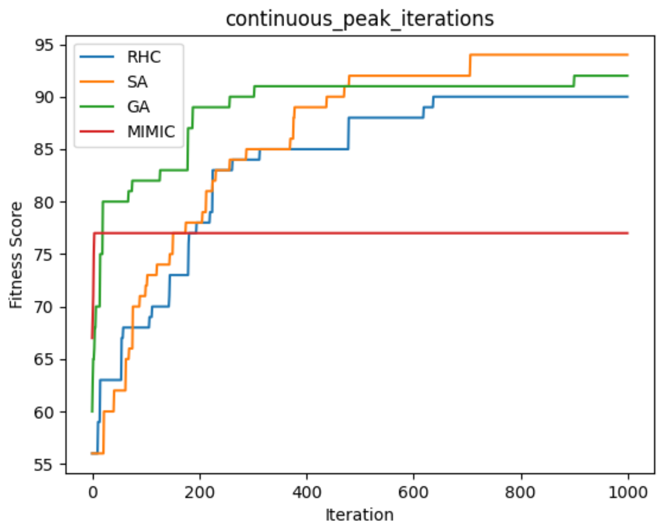
Randomized Hill Climbing and Simulated Annealing both reached one of the two local optima (all 0's in this case). On the other hand, MIMIC algorithm shines in this problem. MIMIC not only achieved the best fitness score but also converged faster than the rest of the 3 algorithms. That is because during each iteration, MIMIC uses the top-performing candidates within the population (in our case, population of 100) to revise probability distribution to "favor" sampling (started as uniform in the first iteration) top-performing candidates in the next iteration, effectively increasing the “fitness” of population during each iteration. In updating the probability distribution, it also takes into account of the “conditional dependency” among the feature space. Therefore, MIMIC performs the best in the problems where the underlying “structure” represented by the feature space is important. And for the Four Peak problem, the underlying “structure” does matter a lot, for example, structure that starts with long string of 1’s and ends with long string of 0’s, or structure that starts with long string of 0’s and ends with long string of 1’s will get high points. On the other hand, Randomized Hill Climbing and Simulated Annealing treats each feature within the feature space as independent from each other.

Optimization Problem 2: Continuous Peak

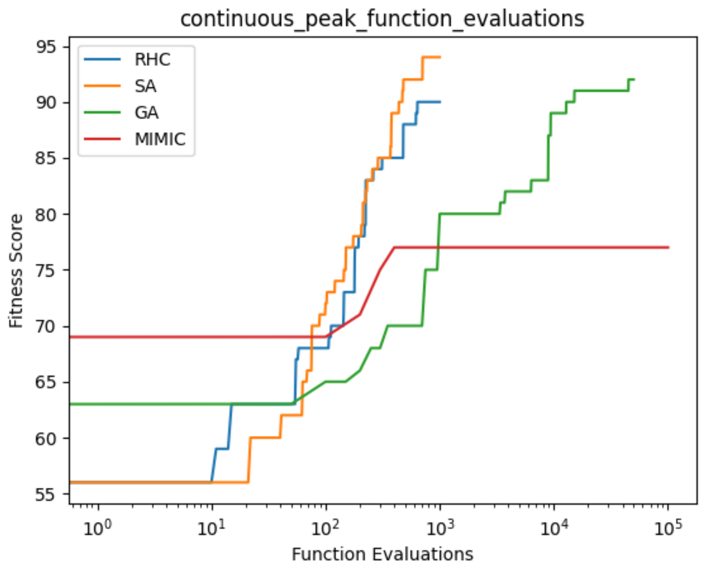
I would like to use Continuous Peak to highlight the optimization technique of **Simulated Annealing**. Fitness for Continuous Peak is defined as: given n-dimensional state vector x where each dimension could be 0 or 1, and given parameter T, then Fitness(x, T) = max(max_run(0, x), max_run (1, x)) + R(x, T) where max_run (b, x) is the length of the maximum run of b’s in x, and R(x, T) = n, if max_run(0, x) > T and max_run(1, x) > T, otherwise R(x, T) = 0.

The problem length that’s set for the Continuous Peak problem is 50. Different from the Four Peak problem above, Continuous Peak is less stringent on the underlying “structure” of the candidate, as the string of 1’s or 0’s can appear anywhere within a bit-string candidate, they do not need to be in the head or tail required by Four Peak.

Below graph shows the fitness score for each of the four optimization algorithms over *iteration*:



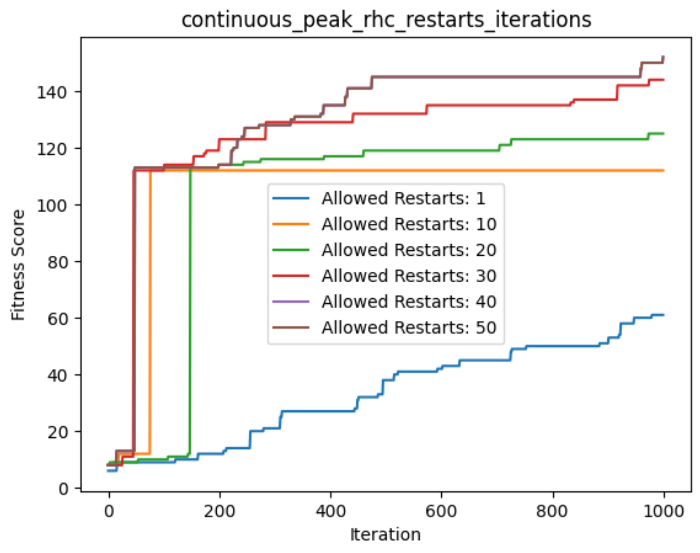
Below graph shows the fitness score for each of the four optimization algorithms over *function evaluations* (Genetic Algorithm and MIMIC naturally require more function evaluation as they need to assess each of the 50 candidates in the population in each iteration):



Simulated Annealing performs the best not only in fitness score but also requires the least amount of iterations and function evaluations to achieve the highest fitness score. Simulated Annealing allows the algorithm to "explore" the parts of the feature space that has lower fitness score than its neighbors, thereby to increase the probability of escaping a local optimum. Exponential Decay and Geometric Decay are the two popular decay methods that controls the chance of "exploring" low fitness value areas. I tried both of the decay methods and they do not seem to matter much in this specific problem.

As I inspect the mlrose code for finding the "next neighbor" for Simulated Annealing and Randomized Optimization, I found out that it does *not* keep track of the neighbors that it has visited in the past. Each time when it needs to find a neighbor, it randomly changes one feature in the feature vector (in our case, a feature vector of 50 1's or 0's) either from 1 to 0, or 0 to 1, and then assess this chosen neighbor. However, this chosen neighbor might have been chosen and assessed many times in the past. Of course, if the algorithm starts to track the neighbors that it has visited, it would increase the memory usage, but it would be an interesting topic to investigate in terms of managing the tradeoff between memory vs efficiency in finding a more fit neighbor.

I'd also like to use the Continuous Peak problem to showcase the power of restarts in Randomized Hill Climbing. For Continuous Peak problem with length of 100, I plotted the fitness curve for Randomized Hill Climbing with 1, or 10, or 20, or 30, or 40, or 50 restarts. See below result:

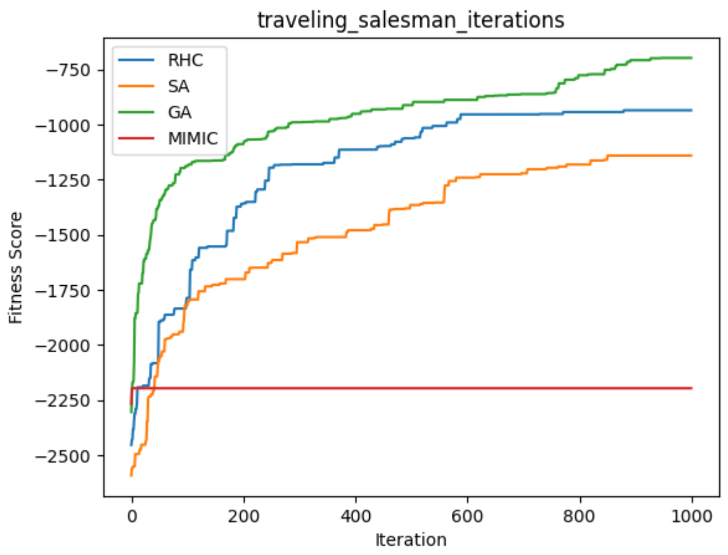


As shown, for this specific problem, high restarts number allows Randomized Hill Climbing to find better fitness score. Of course, obviously, high restarts number also requires more time to compute.

Optimization Problem 3: Travelling Salesman

I would like to use Travelling Salesman to highlight the optimization technique of **Genetic Algorithm**. For this specific Travelling Salesman, I set the number of cities to be 50. Each city defined by is a coordinate, and the optimization algorithm needs to "travel" through each of the 50 cities exactly once. The maximization function equals the negative of the distance travelled (Euclidean distance between coordinates), therefore, the lower the distance, the higher the negative of the distance value.

Below graph shows the performance of each of the four optimization algorithms:



As shown, Genetic Algorithm is not only able to achieve the highest fitness score (low distance), but also increase the fitness score much more rapidly in the first 200 iterations. This can be attributed to Genetic Algorithm's underlying algorithm to smartly 'reproduce' a candidate. First, let's discuss Simulated Annealing and Randomized Hill Climbing's method in picking a neighbor. They simply, by chance, randomly swap any two cities, to see the if the fitness score increases. However, the fitness score depends on the whole order sequence of cities (in our case of 50 cities). On the other hand, Genetic Algorithm pick two parents (parents drawn according to a distribution in proportion to parents' fitness score, therefore, more fit, more likely to become parents), then pick a random point *n* (*n*th city, where n is from 0 to 50 in our case) in parent_1, and parent_1[0:n] becomes the first *n* cities to visit for the child, then child[n:50] is filled by the cities that have not been visited by the child, according to how they were ordered in parent_2. Therefore, a "good" sequence of cities (first *n* ordered cities of parent_1 and the rest of *50-n* ordered cities from parent_2) to visit is more likely to be preserved or passed down from parents to child.

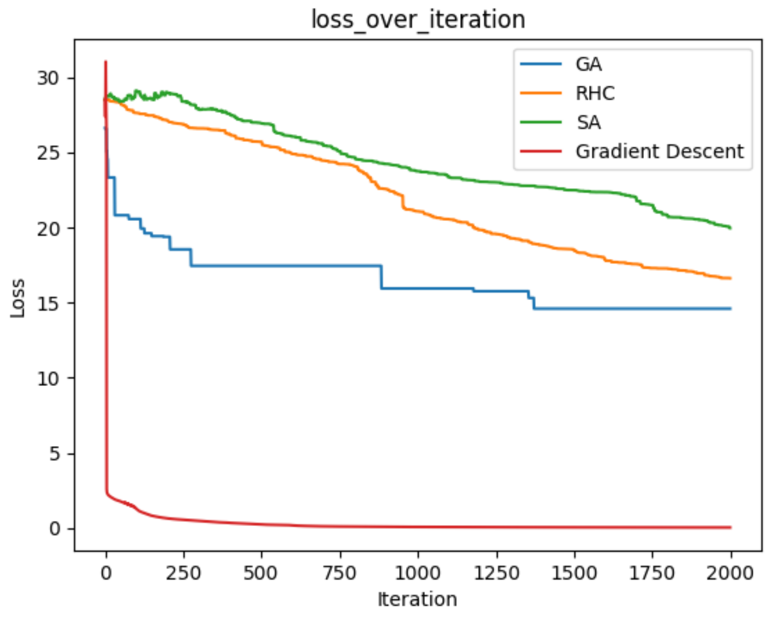
Part 2: Neural Network applied with Randomized Hill Climbing, Simulated Annealing, Genetic Algorithm, and Gradient Descent

I used the MNIST dataset that's used for assignment 1. The neural network architecture with Gradient Descent tuned in assignment 1 includes 1 layer of 90 hidden nodes, combined with RELU activation function, trained with scikit-learn library. I kept the same neural network architecture as assignment 1's for each of the four optimizations for this assigment: Gradient Descent, Randomized Hill Climbing, Simulated Annealing, and Genetic Algorithm.

Gradient Descent: I used scikit-learn in assignment 1's neural network training with gradient descent. Since I am using mlrose's neural network for this assignment, I decided to retune the learning rate and max_iteration parameter a little in order to get to the ~98% training accuracy achieved in assignement 1.

For Randomized Hill Climbing, Simulated Annealing, and Genetic Algorithms, I clipped the weight to be in between -1 and 1, in order to prevent vanishing gradient problem. I also significantly increased the learning rate to 1 from Gradient Descent's 0.0001 learning rate. Reasonings for these adjustments explained later.

Below graph demonstrate the cross entropy loss through each iteration for each of the four optimizations:

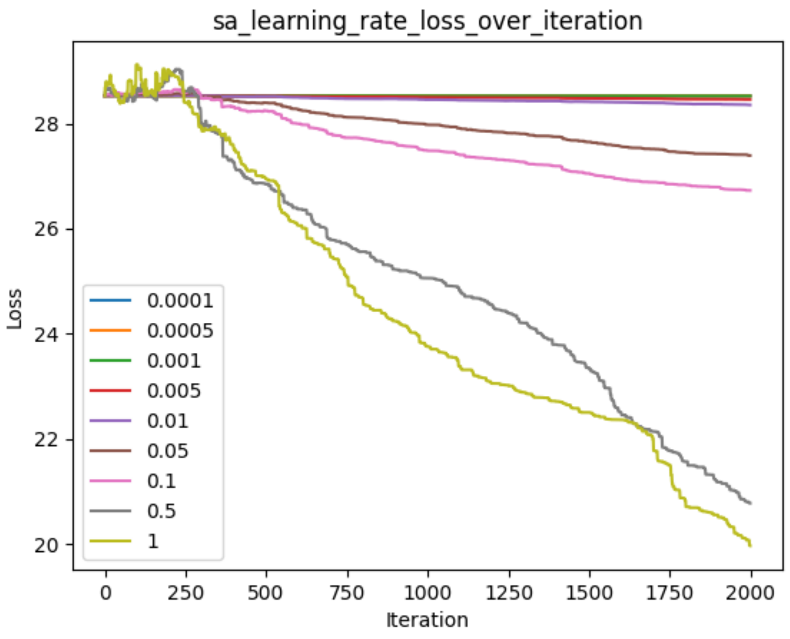


Not surprisingly, Gradient descent is able to rapidly reduce the loss through the first couple of hundreds of iterations. On the other hand, Simulated Annealing and Random Hill Climbing had much slower rate of reducing the loss.

This makes sense because, for Gradient descent, for each iteration, *every* single weight (there are about ~7000 weights for our neural network architecture) has a chance to adjust itself to decrease the loss function. On the other hand, for Simulated Annealing and Random Hill Climbing, for each iteration, they pick one neighbor that has only *one* weight different from the existing weights. Therefore, for each iteration, only 1 weight at a maximum is changed to reduce the loss function.

Hence, if a neural network architecture allows for 2000 iterations, then Gradient Descent gives each weight 2000 times to adjust itself to reduce the loss function. However, for Simulated Annealing and Randomized Hill Climbing, given 2000 iterations limit and ~7000 weights, each weight, on average, has only 30% chance (calculated by 2000 divides 7000) to adjust only *once* to reduce the loss function, through the entire duration of training! Therefore, to partially compensate for this disadvantage, the "step size" (meaning the learning rate) for each weight adjustment has to significantly increase from Gradient Descent's learning rate of 0.0001.

Below chart demonstrates the loss over iteration for different learning rate, using Simulated Annealing:



As shown, the bigger the learning rate (up to 1), the more quickly the loss decreases. However, the bigger "step-size" caused by higher learning rate introduces the higher chance of "over-stepping" a local optimum. The large "step-size" will also encourage large weight value which induces the problem of vanishing gradient. Therefore, I clipped each weight to be in between -1 and 1.

This learning rate issue does not affect Genetic Algorithm, however. Genetic Algorithm produces the next candidate(s) using uniform or single point crossover between two parent candidates, who are more likely to be drown if their 'fitness' is higher (meaning loss is lower). For our specific case, mlrose uses single point crossover. Although Genetic Algorithm performs better in reducing the loss function than Simulated Annealing and Randomized Hill Climbing, it's still nowhere near Gradient Descent's capability reducing the loss function.

Reference

[1] Jeremy S. De Bonet, Charles L. Isbell, Jr., Paul Viola MIMIC: Finding Optima by Estimating Probability Densities