Part 1: Training Neural Network with Randomized Optimization

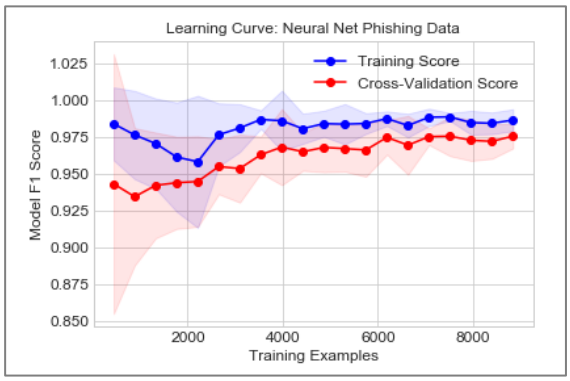
Data and Backpropagation Recap

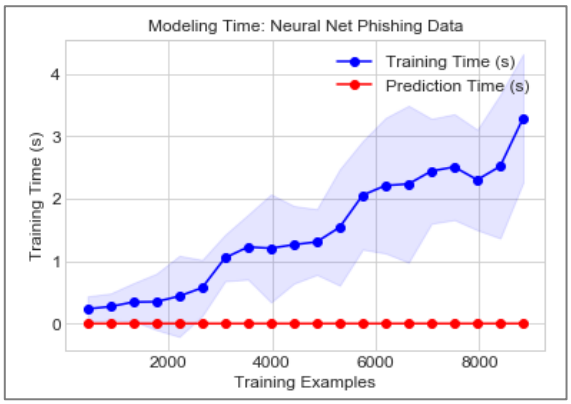
Phishing Websites – available at OpenML.org (https://www.openml.org/d/4534)

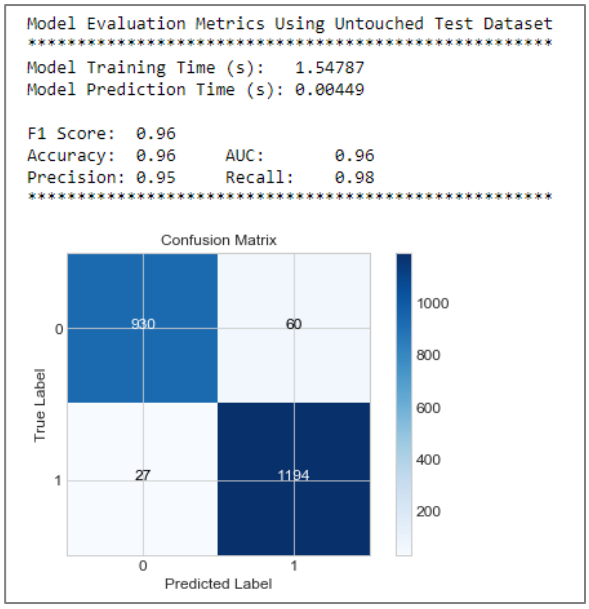
The phishing websites dataset is a compilation of over 11,000 real websites which are labeled as 'legitimate' or 'illegitimate' (spam, untrustworthy, etc.). The binary classification task is to predict which websites are illegitimate such that a user can be notified of the sites' potentially harmful status. The target variable is relatively balanced (55% illegitimate observations) and is accompanied by 30 descriptive features which give information about the website appearance, behavior, URL, and more. All 30 predictors are categorical and 22 of them are binary. I used one-hot encoding to preprocess the dataset.

Based on Assignment 1, I selected my final neural network to have 46 input nodes (one for each feature after one-hot encoding), a single hidden layer of 50 nodes, and 1 output node. With 46 binary input features, the optimization methods are trying to maximize the fitness function of classification accuracy by setting the continuous weight values in the neural network.

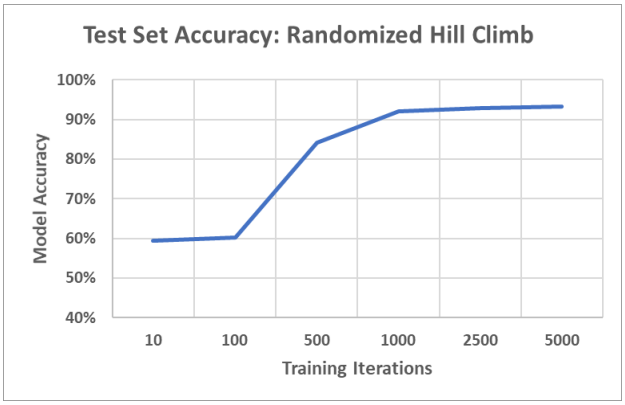
The learning curve as well as training and prediction times from the backpropagation network are shown below as reference before we explore randomized optimization methods on this same dataset. The final backprop model trained in just over 3 seconds and correctly classified 96% of examples in the test set.

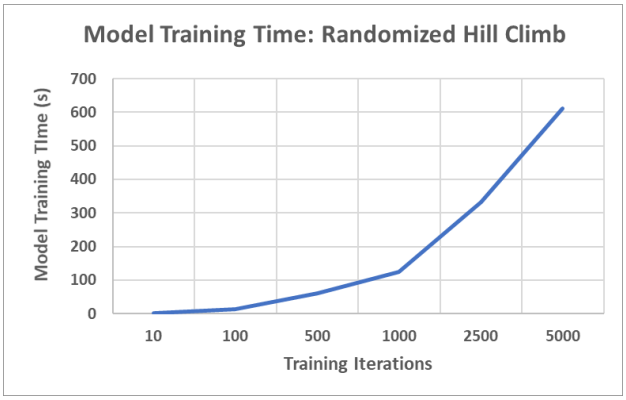






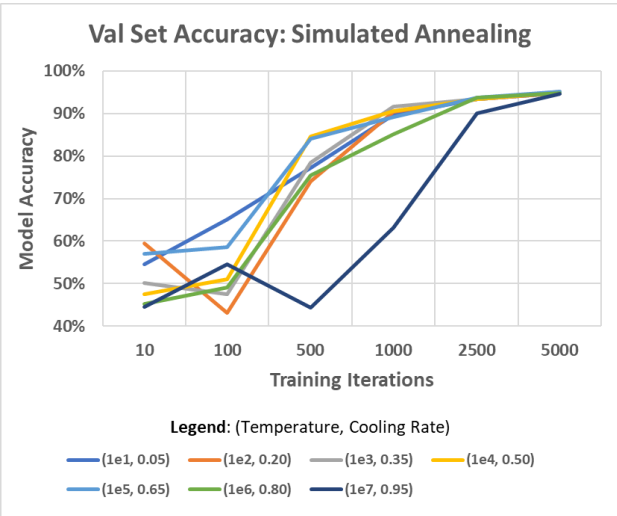
Randomized Hill Climb – Tuning Neural Network Weights

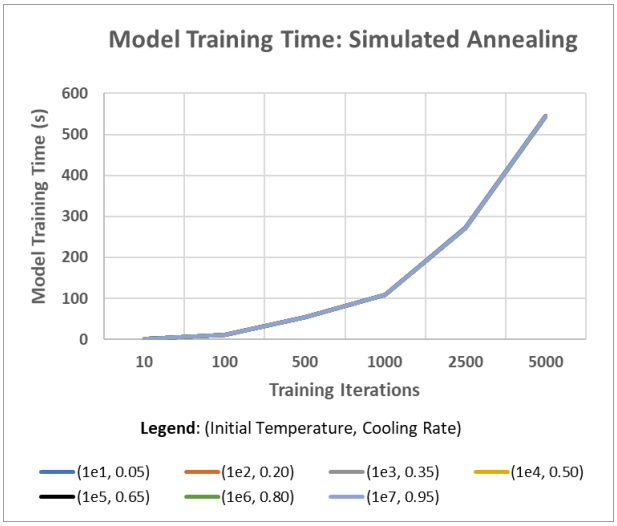




I first used randomized hill climbing (RHC) to determine weights for a neural network. Random hill climbing does not have any hyperparameters to set. Convergence is determined solely based on the starting point and the data/function structure. Random restarts (one per iteration) helps to find global maxima by ensuring that the starting point covers more of the input space. I split the data into 80% for training and 20% for testing (no validation necessary since there are no hyperparameters). We observe test accuracy around 60% for 10 and 100 iterations before the model reaches, and converges to, an accuracy around 92% for 1000 iterations and beyond. This is an example of the hill climbing method getting stuck at local optima when few random restarts are used. The model training time scales linearly with the number of iterations. The tradeoff between model accuracy and computational time is worth noting. There is only a marginal gain in accuracy between 1000 and 5000 iterations (roughly 1%), yet the model training time goes from about 2 minutes to 10 minutes.

Simulated Annealing – Tuning Neural Network Weights



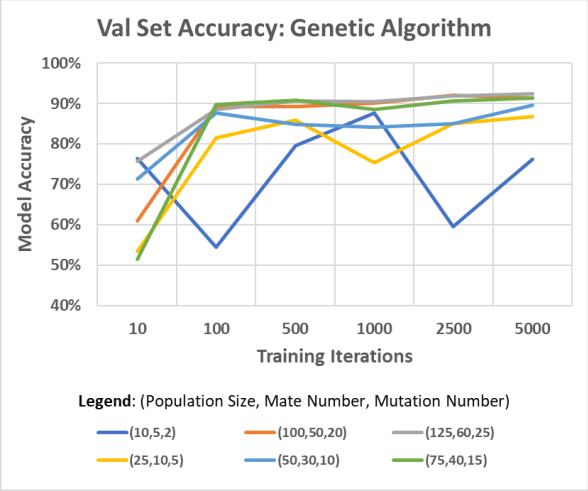


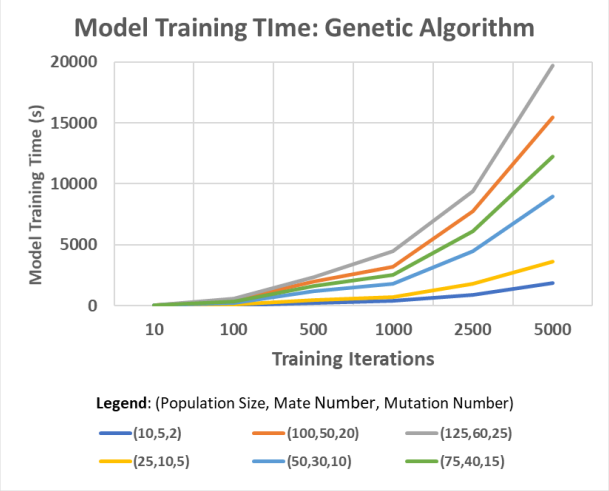
The simulated annealing (SA) method has two hyperparameters to set: initial temperature and cooling rate. Higher starting temperatures emphasize exploration of the function space vs. exploitation. The higher temperature can be viewed as higher energy/randomness, which allows the optimization method

to go "downhill" in order to explore new neighborhoods that might lead to a better optimum. On the contrary, lower starting temperatures limit exploration making the algorithm behave more like hill climbing. I explored seven combinations of these two parameters in training the neural network. I used 60% of the available data to train the models, 20% to generate the validation scores, and I kept a final 20% for later testing.

We observe similar behavior for all validation accuracy curves with the exception of the highest temperature and cooling rate (1e7, 0.95) curve. This model is interesting because it holds a low accuracy for a high number of iterations. This is due to the high temperature causing the model to "jump over" and miss the global extreme. There is too much randomness/exploration in this model. All of the models had the same training times, which were very close to the training time of the RHC method. All of the curves with starting temperatures lower than 1e6 show high accuracies after only 1000 training iterations with only marginal gain for additional iterations.

Genetic Algorithm – Tuning Neural Network Weights





The genetic algorithm (GA) had three hyperparameters to set: population size, the number (or fraction) of individuals to mate, and the number (or fraction) of individuals to randomly mutate. Each iteration represents one generation of the hypothesis population where the most fit hypotheses are chosen to pass on their genes to the next generation through crossover. I used 60% of the available data to train the models, 20% to generate the validation scores, and I kept a final 20% for later testing.

The model training time plot shows us that as we have a more complex model (larger population, more mating, more mutation) the model training time increases. In this case, model training time is a significant concern. The largest model took over 5.5 hours to train for 5000 training iterations! Genetic algorithms are easy to parallelize to reduce the cumbersome effects of long training time. So there isn't an inherent reason to choose a small population size. Yet, for this assignment I did not parallelize, so I'll stick with smaller population sizes in further testing.

We observe similar validation accuracy across models when we have a sufficiently large population size. The accuracy curves reach 90% and stabilize fairly well after only 100 iterations for population sizes of at least 75. For this reason, we would prefer to choose the simplest model, shown by the (75,10) line,

since it has the lowest training time among the curves which have the high, stable accuracy. For population sizes of 10 and 25, we observe more randomness in model accuracy. This is due to the smaller sample sizes not providing enough strength to overcome local optima.

Second Hyperparameter Check and Model Comparison

The previous section showed that there exists a combination of hyperparameters in SA and GA for which the model validation accuracy exceeds 90% by 1000 iterations for SA and by 100 iterations for GA. This is great because the GA method in particular took a long time to train for higher numbers of iterations. 1000 iterations were also enough for the RHC method to find greater than 90% accuracy. In all three algorithms, there was only minimal gain in accuracy for increasing the number of iterations. As a final check on hyperparameters for SA and GA, I played with a new set of parameters for models up to 1000 iterations (using the same 60%/20% split for training and validation). I used the original hyperparameter search as well as this one to determine a "best" model for SA and GA, which are compared to RHC in the next section.

Simulated Annealing	
(Temp, Cooling)	Val Accuracy
(1e3,0.2)	0.851
(1e3,0.4)	0.910
(1e3,0.6)	0.827
(1e4,0.2)	0.845
(1e4,0.4)	0.858
(1e4,0.6)	0.839
(1e5,0.2)	0.845
(1e5,0.4)	0.858
(1e5,0.6)	0.839

Genetic Algorithm	
(Pop, Mate, Mutate)	Val Accuracy
(50,20,1)	0.885
(50,30,1)	0.879
(50,20,5)	0.880
(50,30,5)	0.872

Neural Network Training Method Comparison

As a final means of comparing the RHC, SA, and GA I ran the top models for a final run of 5000 iterations over the previously untouched 20% test set. The model test accuracies and training times are shown below. As in the previous examples, we see that RHC and SA have similar training times while GA is an order of magnitude larger. RHC and SA each reach test accuracies in the low 90s. It is likely possible to get low 90s for the GA as well if hyperparameters are further tuned. Overall, these three optimization models fall short of the neural network trained using backpropagation which had a test set accuracy of 96%. For this dataset there appears to be about a 3% performance difference between backpropagation (which minimizes the total error across the network weights) and the optimization methods (where a fitness function is being maximized). It is expected that the randomized search algorithms would perform worse than backpropagation for training a neural network continuous-valued weights because these methods are just trying to find a "high fitness" function within an infinite hypothesis space.

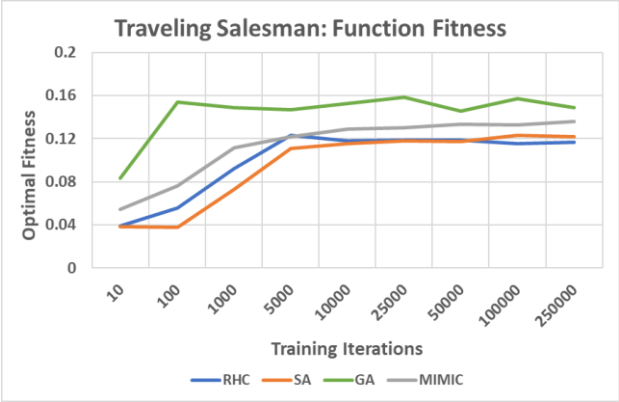
	RHC	SA (1e3, 0.40)	GA (75,25,1)
Test Accuracy	0.930	0.929	0.887
Training Time (s)	585	620	7526

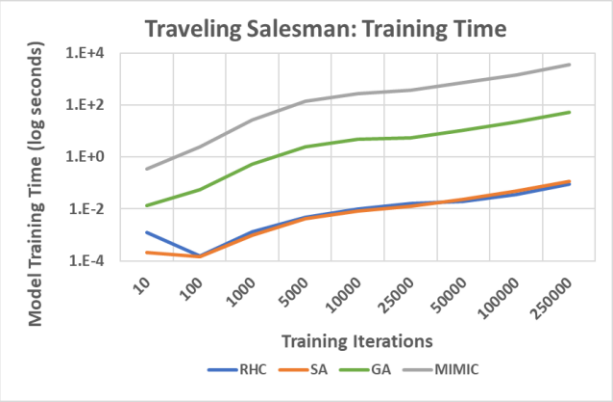
Part 2: Randomized Optimization on Three Toy Problems

In this section, I will present three unique optimization toy problems where the domains are represented by bit strings. Each problem will be solved (the fitness functions maximized) with four randomized optimization methods: RHC, SA, GA, and Mutual Information Maximizing Input Clustering (MIMIC). The MIMIC algorithm has the distinct advantage over the other three methods in that it can track the history of successive iterations. In this, MIMIC is able to utilize structure through conditional dependencies in the input domain space to find the global optima. This also means that MIMIC can be stopped early and it will still produce an output. MIMIC typically converges in fewer iterations than the other methods, but at a cost of longer training times per iteration. We will see how this works in practice.

For each toy problem I will run the optimization method using their ABAGAIL-preset hyperparameters: SA (Temp=1e11, Cooling=-.95), GA(Population=200, Mating=100, Mutation=10), MIMC(Population=200, TopN%=10%). I will evaluate each algorithm's fitness and model training time for {10, 100, 1000, 5000, 10000, 25000, 100000, 250000} iterations. For each iteration level, the reported fitness and training times are the average across 5 separate executions of the models.

Problem Domain 1 – Best with GA: Traveling Salesman





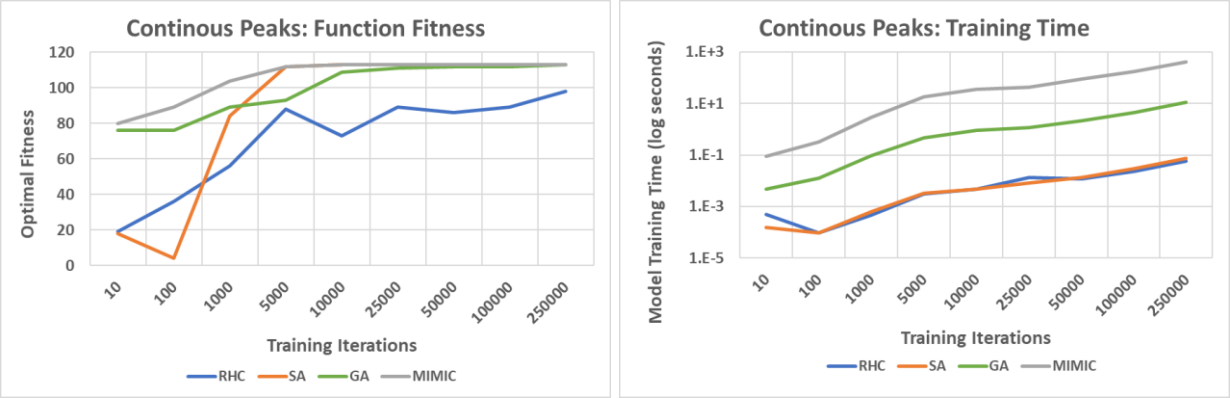
The traveling salesman problem is an optimization problem where the fitness function to maximize involves ensuring that a salesman visits each of his customers exactly once, by using the minimum total travel distance to accomplish it. The fitness function, in terms of a bit string, calculates the distance between customer locations. Smaller distances result in higher fitness values. I used the ABAGAIL-default of N = 50 customers for the salesman to visit. This means that the total hypothesis space contains 50! = 3.064e64 ways in which the salesman could order the visits! This is a great problem for randomized optimization because the hypothesis space is too large to simply enumerate all possible fitness values to find the best one. While we may not find the absolute best travel route with this method, we can find one that is close to the best route in a relatively small number of iterations.

The fitness function plot above shows that RHC, SA, and MIMIC find similar fitness values at each iteration level. These three algorithms have difficulty determining the traits which make one hypothesis (a travel route) better than another. in The GA finds a significantly higher fitness, and it finds it quickly – in terms of both iteration count and wall clock time. The GA fitness value reaches 0.154 after only 100

iterations in a clock time of 0.6s. In this same amount of time RHC and SA can train 250,000 iterations. Yet, the optimal fitness value that RHC and SA find is only 75% of that which GA finds.

A GA approach to the traveling salesman problem works well because of the problem constraints. The problem mandates that each customer be visited once and only once. This means that crossover and mutation techniques within the GA need to abide by this standard. For example, we can't perform mutation where customer A is replaced by customer B. In this case, the salesman wouldn't ever visit A and he would visit B twice. Therefore, mutation and crossover are done in a way that isn't as random as other applications of GA where a "bad gene" can be introduced to the hypothesis population which significantly lowers fitness. For this problem the GA will naturally preserve the best routes and pass that information along to the next generation. GA accomplishes this well because this is not an instance-based problem, rather it is distribution-based. The more fittest hypotheses have the highest chance of being selected for mating. In an evolutionary view, successive generations make improvements very quickly. That's why we see the GA find a high fitness value after just 100 iterations, and this value is roughly the same even up to 250,000 iterations. The GA could be improved even further by exploring a hyperparameter search across the population size, mating size, and mutation size. Although I suspect only minimal improvements would be made even at the best hyperparameter settings since the current algorithm already finds a high fitness value quickly.

Problem Domain 2 – Best with SA: Continuous Peaks



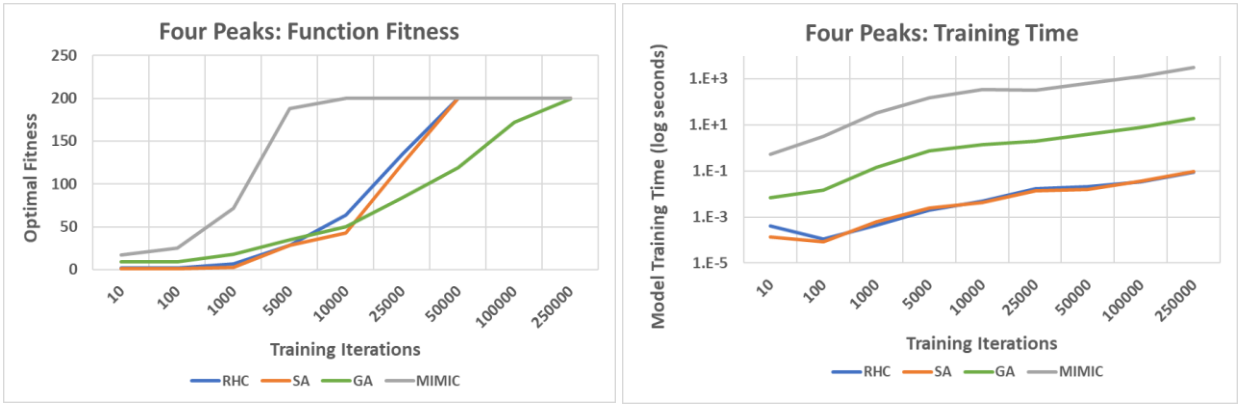
The continuous peaks problem has many local optima which to traverse in an attempt to find the global maximum. The fitness function is a bit string of length N = 60 where a fitness reward is given for strings with at least T = 6 consecutive zeros or ones anywhere within the string. This is a great problem for randomized optimization because there are 2^{60} = 1.153e18 total possible hypotheses to search over. It isn't practical to evaluate all of these hypotheses, so we use the search algorithms to quickly find a hypothesis with a high fitness value.

The first plot above shows that RHC generally finds a better fitness value as more iterations are added. This makes sense because the function space in continuous peaks has many local optima where RHC can get stuck. It takes more iterations for RHC to get stuck again, but at a higher fitness value than its previous sticking point. We observe that GA starts with a relatively high fitness value for only 10 iterations, but it is slow to improve as iterations are added. In this case, there is only marginal improvement from one generation to the next. The introduction of mutation makes it difficult for the GA to maintain the

consecutive sequence of zeros and ones which is what leads to it earning the fitness bonus. We note that MIMIC performs well in finding high fitness in few iterations. MIMIC is able to learn how neighboring bit values are conditionally dependent in giving higher fitness. This is similar to how MIMIC detects structure in datasets by learning which inputs are conditionally dependent.

MIMIC could be seen as the best algorithm for the continuous peaks problem, until you introduce the training time plot. We can see that MIMIC reaches its best fitness value of 113 after 5000 iterations. However, SA also reaches the same fitness in 5000 iterations. With equivalent fitness values, it is natural to say that the better algorithm for this problem is the one which takes less computation time. SA is able to reach the same fitness value as MIMIC in less than 1/5000th of the time! SA works so well on this problem because of its exploration property. SA is able to navigate the many local optima to find a high fitness value because it isn't as susceptible to getting stuck at local optima as RHC. We do observe SA getting stuck at local optima for very low iteration counts, but SA quickly finds a better fitness value when more iterations are added. SA improves much more quickly than RHC since it can utilize the exploration property whereas RHC relies on random restarts and getting lucky to find high fitness values. Further improvements on the SA method could be make by tuning the initial temperature and cooling parameter. The end result would be that the SA method finds the optimal value of 113 in a lower number of iterations.

Problem Domain 3 – Best for MIMIC: Four Peaks



I investigated the four peaks toy problem by defining function fitness over a bit string with N = 200 bits. The fitness function gets a bonus if the first T = 40 bits of the string are all ones, or if the last 40 bits are all zeros. A bit string of length 200 makes the size of the hypothesis space 2^{100} = 1.268e30. Again, too many to enumerate through, so we use randomized optimization methods to find a high fitness value.

The first plot above shows that GA is very slow (requires many iterations) to reach a fitness value of 200. This is a difficult problem for GA because any mutation in the first or last 40 bits of the bit string can cause the hypothesis to lose the fitness bonus. Therefore, we require many iterations of GA to see high fitness. RHC and SA performance is roughly equivalent. These methods easily get stuck at local optima. We deduce that the local optima in this landscape are surrounded by large plateaus because the exploration property of SA isn't enough to traverse the plateaus to find fitness values better than RHC. The only thing that helps the hill climb methods is adding more iterations.

The function fitness plot clearly shows that MIMIC is advantageous in this toy problem. MIMIC finds a fitness of 200 in 10,000 iterations. It even finds a high fitness of 188 after only 5000 iterations. MIMIC does take the longest to train, though. The bit strings in this problem have a clear structure where the leading and trailing 40 bits are related to each other. They must match in order to get the fitness bonus. MIMIC takes advantage of the fact that the leading and trailing bits are conditionally dependent. MIMIC can utilize this structure to provide some navigational insight to the optimization search through the hypothesis space. It is interesting that MIMIC is able to maximize the fitness value (find the global optima) in 10,000 iterations – likely fewer if I add some data points to the first plot between 5,000 and 10,000. 10,000 iterations to search through 1.268e30 possible hypotheses! It may be possible for MIMIC to find the global optima in even fewer iterations if we tune the hyperparameters of population size and the cutoff threshold for "top n% fittest" hypotheses.

Model Comparison

These three toy problems show that there is not a one size fits all randomized search method that will perform the best on a given problem. This is the no free lunch theorem in action. Rather, the problem's fitness function definition and constraints lead to certain problems being better suited for different applications. A key difference is the difference between problems suited for instance-based methods (RHC, SA) and distribution-based methods (GA, MIMIC). Additionally, it is important to note the computational complexity of these algorithms. We saw one example (Continuous Peaks) where at first glance it appeared that MIMIC may perform best. It was only upon investigating the model training times that we observed SA reached the same performance as MIMIC in significantly less computational time.