Alan Li
Project 2
CS4641

Randomized Optimization Analysis

Optimization Parameters

Before analyzing the performance of each optimization algorithm, I will define the specific implementations that will be used on each problem.

RHC:
- neighbor function that flips one bit at random

SA:
- neighbor function that flips one bit at random
- Initial temperature of 1×10^{11}
- Cooling rate of 0.95

GA:
- Population size of 200
- Mating number of 100
- Mutation number of 10
- Crossover function that chooses a random index *i*, and returns a new vector with bits [1,*i*] from the first parent and the remaining bits from the second parent

MIMIC:
- 200 samples taken each iteration
- 20 samples kept
- Use of a dependency tree to represent probability distribution

Optimization Problems

Four Peaks

The first optimization problem I chose to illustrate the strength of genetic algorithms was the Four Peaks problem. This problem takes an N-dimensional vector, X and a parameter T, and has the following evaluation function:

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

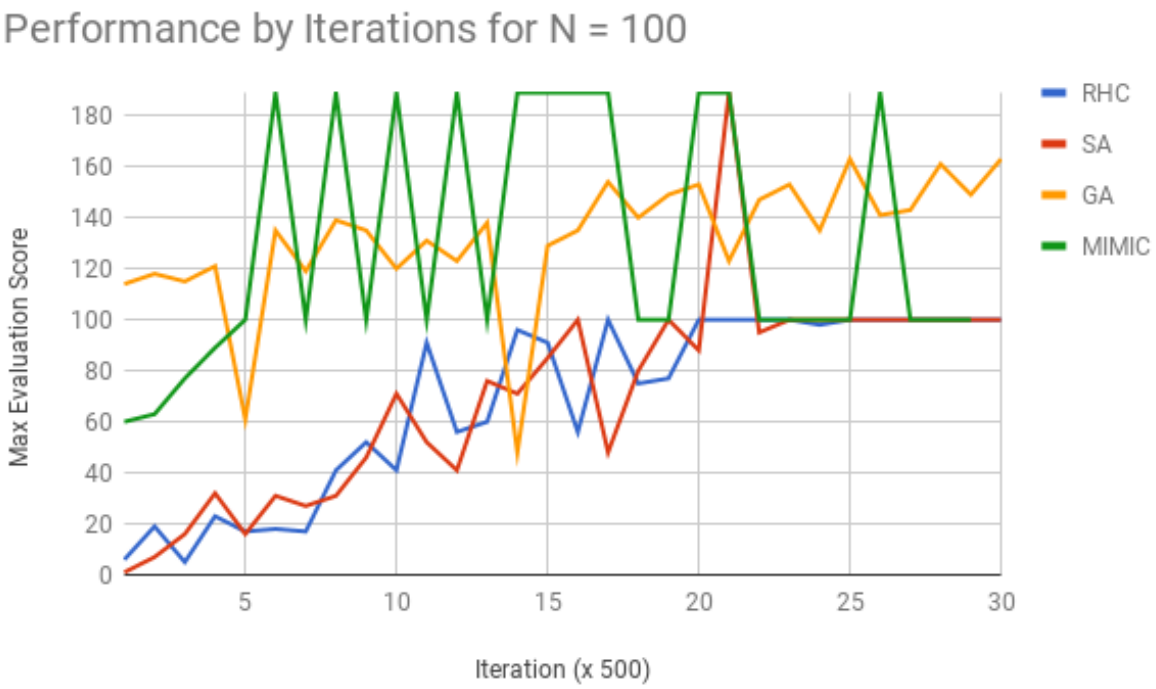
where

$$tail(b,\vec{X}) = \text{number of trailing } b\text{'s in } \vec{X}$$

$$head(b,\vec{X}) = \text{number of leading } b\text{'s in } \vec{X}$$

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

There are four optima in this problem: two of which are a vector full of either 1's or 0's, in which your evaluation is N. These two optima are local. The two global optima are strings such that either the first N + 1 bits are 1's and the rest are 0's or the last N + 1 bits are 0's and the rest are 1's. These two optima evaluate to 2N – (T + 1). The problem is specifically designed to test an algorithm's ability to recognize the underlying structure and break through the local optima. The following data is for N = 100, T = 10:



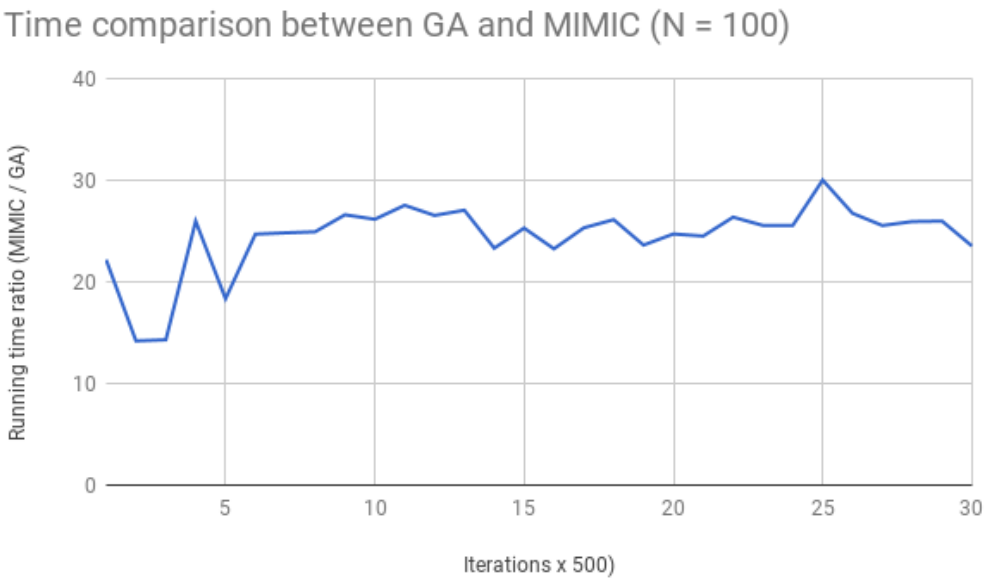
Above we have the best evaluations score as a function of the number of iterations each algorithm runs. Here we note that RHC is unable to capture the underlying structure of the problem. This poor performance is due to the greedy nature of RHC, which has no notion of history and only keeps track of single instance. Unless it is extremely lucky, there is no way RHC can cross the $R(\vec{X}, T)$ threshold, because its neighbor function only changes one bit at a time. We can see from the graph that RHC continues to improve until hitting the local max of 100.

SA has similar performance to RHC. This can be explained by the same reasons we used to explain RHC: SA is still inherently greedy, and only keeps track of one instance. However, from the graph, we notice that in one experiment, SA crossed the $R(\vec{X}, T)$ threshold, and converged to the global max. We note that this is technically possible due to SA's ability to make random decisions, but is highly unlikely. This likelihood should also decrease as the problem size increases, because the number of consecutive numbers required to cross T increases. According to the evaluation function, creating trailing 0's or leading 1's is less optimal than just filling the vector with a single number; thus, to cross the threshold, SA would have to, by chance, make at most T + 1 random decisions. This is extremely unlikely for large values of T. A way to

improve the chances of breaking past the local max would to make a bolder neighbor function. Maybe instead of flipping a single bit, we could set 5 consecutive bits to either 0 or 1. This change could decrease the number of random walks required to break the max.

From the graph, we see that genetic algorithms performed extremely well. Even at just 500 iterations, it broke past the local max and continued to converge towards the global max. This particular GA could break past the local max due to its clever crossover function. We note that we only need two vectors that have converged to the local max—a vector full of 1's and a vector full of 0's—to break past the local max. If we breed these two instances, we have a $1 - \frac{2T}{N}$ chance of creating a successor that has an evaluation function greater than the local max. However, we must realize that not all genetic algorithms are not necessarily good for this problem, but rather it is the clever crossover function that lets the algorithm discover the underlying structure. Let us suppose another crossover function: creating a new vector with all its even bits coming from one parent, and all its odd bits coming from another. Not only is this crossover function suboptimal, it is quite possibly the worst crossover function for this problem. Imagine two vectors that have reached the local max: a vector full of 1's and 0's. Breeding these two would result in a vector that has an evaluation value of 1—the minimum of the function. Breeding two vectors that are full of the same number will only result in the same vector. In other words, we must recognize that our GA performed well in this problem due to our domain knowledge. This makes sense, as GAs typically are useful for expressing domain knowledge by giving us the ability to hand-author specialized mutation and crossover functions.

MIMIC also performed extremely well on this problem. We can attribute this performance to MIMIC's ability to discern history and to communicate information about the underlying structure between iterations, thus making the max of the evaluation function more and more likely. We also note that it took much fewer iterations than the other algorithms to reach its max. However, we must note that, depending on each iteration, MIMIC could still get stuck at the local optima, whereas, GAs could reliably break past the local optima. There are several reasons why MIMIC could have gotten stuck at the local optima. First, it generates new samples based on the previous samples. If those samples happened to all converge to a local max, then the new samples would also converge, and there would be no way to propagate past it. Second, if you get unlucky with your sampling, your distribution would be skewed, and MIMIC could get stuck at some local optima. That being said, MIMIC hit the global max in some cases, whereas GAs did not. One could argue that MIMIC in fact performed better than GAs as a result. While this may be true depending on the results you want, we must note MIMICs running time in comparison with GAs:



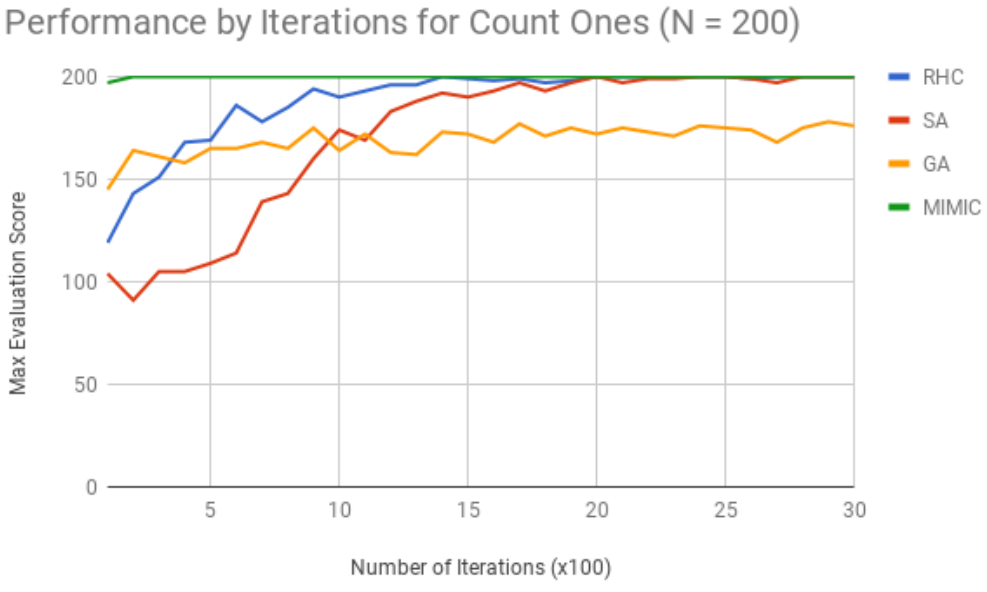
As indicated by the data above, MIMIC consistently took over 20 times longer to run. A reason for this time trade-off is because MIMIC must construct dependency trees and compute with conditional probabilities. While MIMIC excels at requiring fewer evaluations of C(x)—the cost function—, C(x) is very cheap to calculate in this problem. Thus MIMIC seems out of place; rather, there are problems where the strengths of MIMIC shine more. Of course, if we really care about converging to the global max or minimizing the number of iterations we need, then MIMIC outperforms the other algorithms. I did not include the run-times for RHC or SA because they were trivial compared to GA and MIMIC.

Count Ones

The second problem I chose, which will highlight the strength of SA, is the Count Ones problem. The problem is simple:

$$f(\vec{X}) = \text{ The number of 1's in X}$$

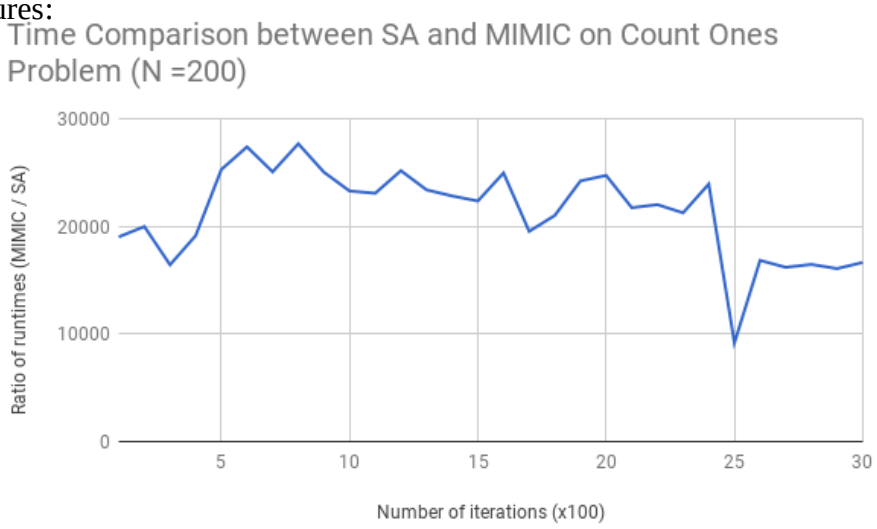
where $\vec{X}$ is an N-dimensional vector. The function has a single optimum—a vector full of 1's—which also happens to be the global optimum. Even before testing, we can hypothesize that a greedy algorithm would perform quite well on this problem:

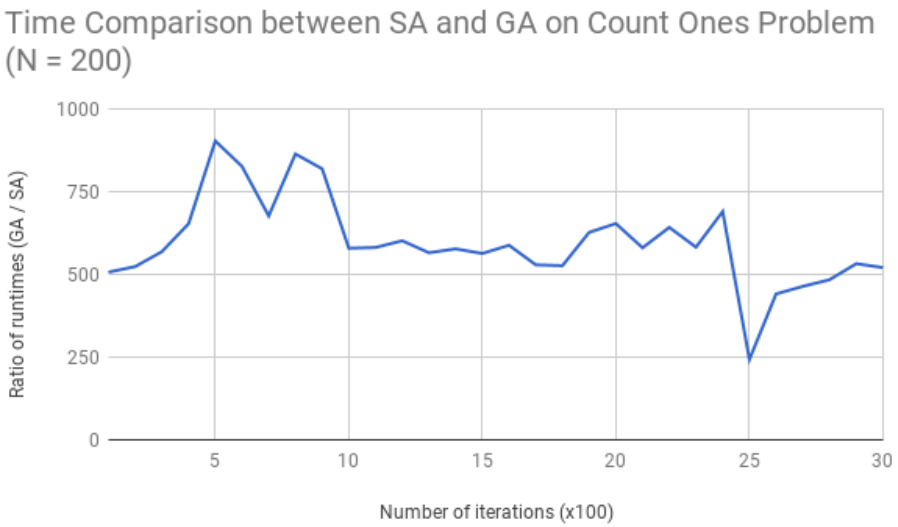


The data only supports our hypothesis. Both greedy algorithms (RHC and SA) quickly converged to the global max, whereas the GA was unable to converge by the experiment's end. A reason for the great performance of both RHC and SA is that they only keep track of one element, and immediately switch upon finding a better neighbor. What usually quells greedy algorithms is their inability to break out of local optima; however, this problem does not have such a structure, as the only optimum is the global optimum. As an extension to this argument, we can hypothesize that SA did slightly worse than RHC during the lower iterations because it made random walks trying to escape a local optimum which did not exist. Nevertheless, my implementation of SA made fewer random walks as the number iterations in increased. This lead to SA performing about the same as RHC during the higher iterations.

From the data, we notice that the GA started better than both RHC and SA, but was unable to reach the global max. One possible reason for this performance is that in this problem the crossover function did not explicitly help the population converge to the max. For example, take two instances with one being full of 1's in the first half, and the other being full of 1's in the second half. The crossover function could either create a child with a maximum value or a child with 0 value. In general, the crossover function technically combines two parents, but does not necessarily create a "better" child. The data supports this hypothesis, as we see the performance plateau even as the number of iterations increases. We also note that our mutation function creates a worse instance half the time. A possible way to improve the performance is to increase the population size, create a more drastic mutation function, or to create a better crossover function. Again, GAs are strong due to their ability to represent domain knowledge. A good cross over function for this problem would be to bitwise OR the two parents.

Lastly, we look at MIMIC, which seems to straightaway converge to the global max. This makes sense because MIMIC is supposed to find the max with fewer iterations due to its leverage of history and persistent storage of data. The problem with using MIMIC for a problem like the Count Ones problem is that it does not take advantage of MIMIC's time trade-off. Again, calculating C(x) is easy for this problem, so in the end MIMIC will have terrible run-time performance in comparison to the other algorithms due to its construction of supporting data structures:





For large N, it would not make sense to use MIMIC. Even though MIMIC does find the max with fewer iterations, it still takes less time to find the max with SA. GA not only takes longer, but also performs worse than the other algorithm. In reality, a problem this simple can be solved best by using RHC or SA.

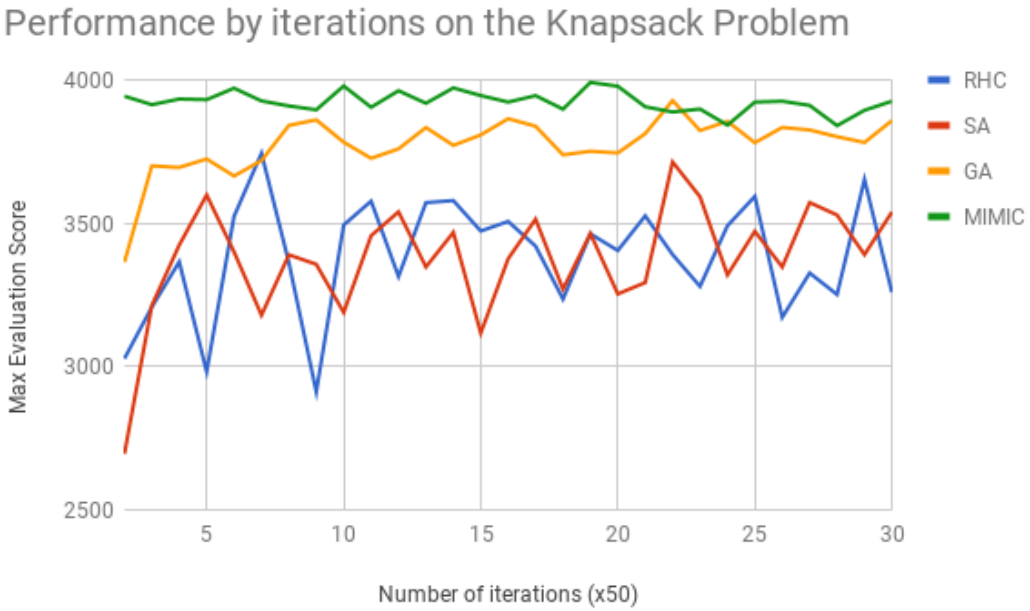
Knapsack Problem

The last problem, which will try to elucidate the benefits of MIMIC, is the knapsack problem. Intuitively, the knapsack problem is structured as having several items, each with a set amount of value, weight, and supply. Our goal is to try and maximize the value of our "knapsack" by picking the optimal combination of items. What makes the problem difficult is that we have a limit to the weight we can carry (this is known as the bounded knapsack problem). To put it more formally, given n items, numbered 1 to n, each with a value v_i and a weight w_i, a max weight of W and a stock of c of each item,

$$\text{maximize} \sum_{i=1}^{n} v_i x_i$$

$$\text{subject to} \sum_{i=1}^{n} w_i x_i \leq W \text{ and } 0 \leq x_i \leq c$$

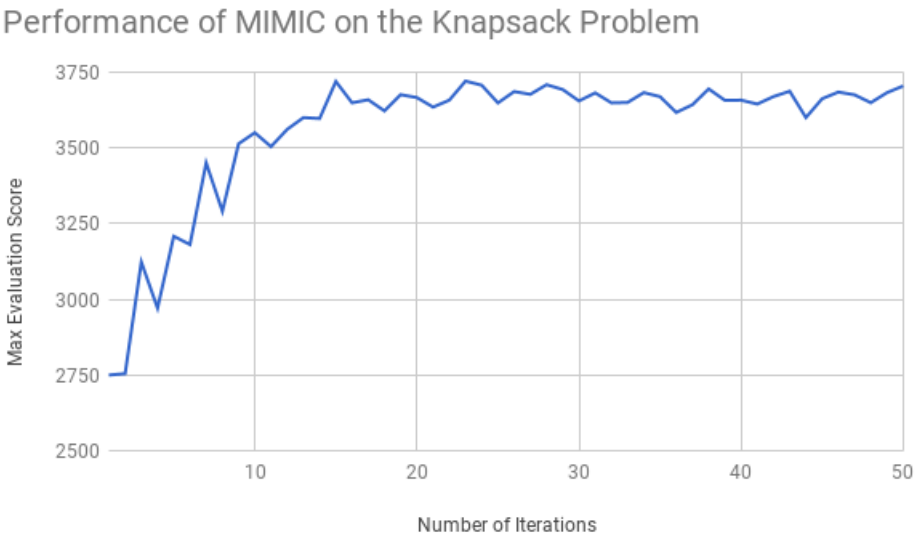
where x_i is the number of copies of an item being put in the knapsack. My implementation of this problem randomly initialized the values and weights of 40 items, each with 4 copies. Thus, the optimal solution of this problem is arbitrary. In addition, the structure of this problem is also unknown. Depending on how the items are initialized, there may be several local optima. Given the somewhat undefined structure of the problem, we can expect greedy algorithms such as RHC and SA to be easily tricked here:



From the data, we can clearly see that MIMIC scored consistently higher than any other algorithm. We must also note that the GA also scored consistently higher than the greedy algorithms, which both not only performed poorly, but also lacked consistency in their results.

We can attribute the poor and inconsistent results of both RHC and SA to their greedy nature. It is evident that both these algorithms were unable to break past the problem's local optima, often getting stuck at one of the many peaks below. This can also explain the inconsistency of the results, because depending on how the instance was initialized, it would get stuck at a different peak each experiment. We also note that the results of both RHC and SA do not improve as the number of iterations increases. This further suggests that these algorithms converged to a local optimum. While it is reasonable to expect SA to perform better than RHC due to its random walking, that is not the case. One possible reason is that, like the Four Peaks problem, it would take multiple random walks to break past the local optima, which is highly unlikely.

Both GAs and MIMIC performed comparatively well. One reason why GAs performed well is its ability to capture the underlying structure of the problem with the crossover function. Whereas the greedy algorithms only looked at its neighbors, GAs used a distribution of instances to gain insight to the structure of the problem. MIMIC also performed well for this reason. But instead of using a crossover and mutation function, MIMIC leveraged auxiliary probability models to capture the structure of the problem. An interesting note is that MIMIC seems to have already converged at just 50 iterations. I did an additional experiment to explore MIMIC at even lower iteration numbers:



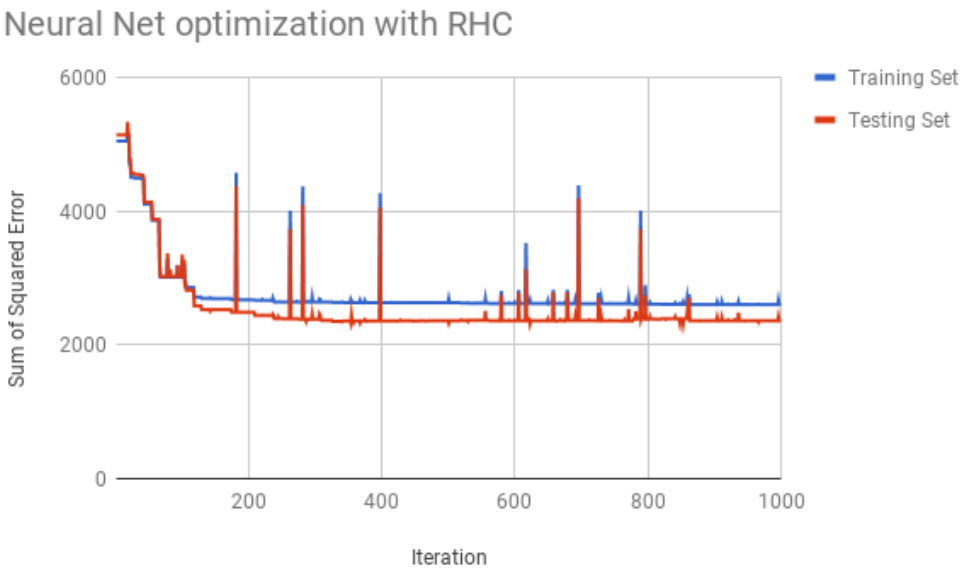
The experiment above suggests that MIMIC can converge to its optimum using about 20 iterations, whereas the greedy algorithms could take 1000 iterations and still perform about 12% worse. In this problem, run-time comparisons are unneeded because no matter how many times you run RHC or SA, they will get stuck at a local optimum. We have already established that GAs will run faster than MIMIC for the same number of iterations, but even so MIMIC performs better than GAs at 50 iterations than GAs do at 1500 iterations. Thus, due to the space constraints of this paper, I will not go into further detail in the running time comparisons.

Neural Net Weight Optimization

The dataset I used for this experiment was the bank dataset from assignment 1. It has 16 features and a binary output, which represents whether a customer will subscribe to a term deposit with the bank. I used the ABAGAIL library to implement each optimization algorithm. Because I used a different library and architecture for my neural net, I also redid the experiment from assignment 1 (using backpropagation). Below are the final training and testing scores for each neural net. Each net consists of a size 16 input layer, a size 5 hidden layer, and an output node. Every algorithm was given a 0.05 error threshold, or 1000 iterations, whichever comes first:

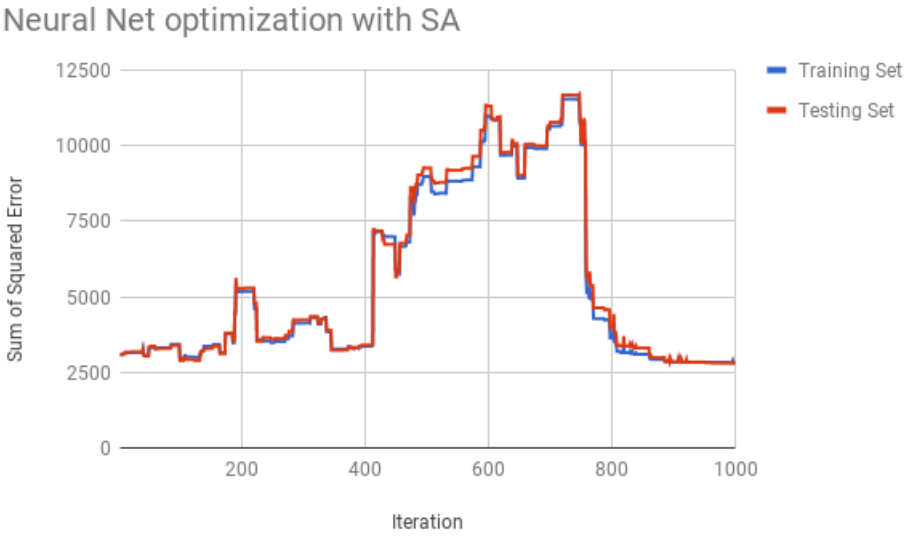
Net Algorithm	Training Score	Testing Score	Optimization/Training Time	Testing Time
Backprop	79.557	78.467	0.3	0.04
RHC	82.935	82.233	126.306	0.16
SA	79.256	77.934	122.291	0.14
GA	80.345	80.268	3150.202	0.134

RHC



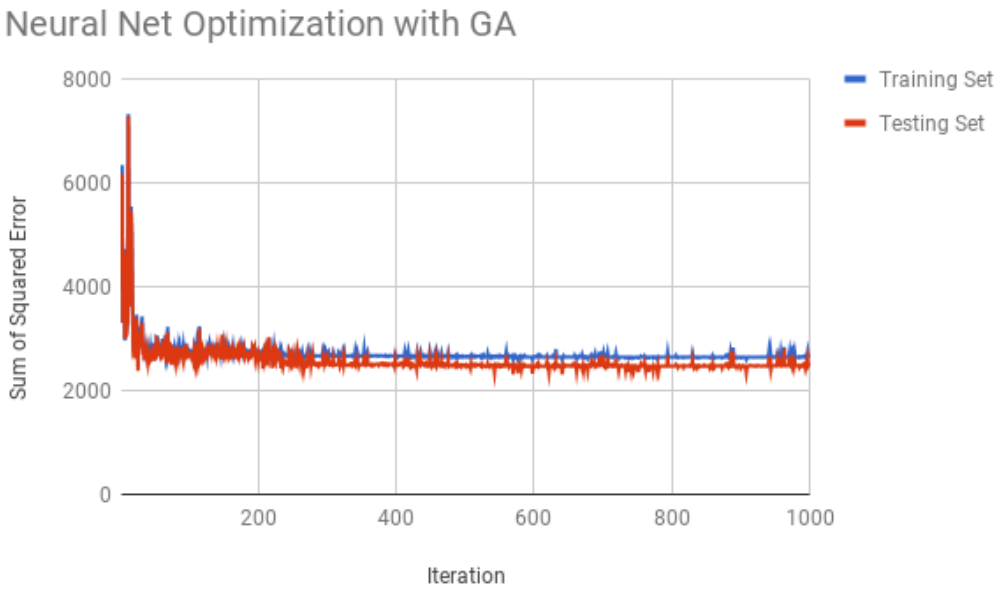
Above are the training and testing curves for a neural net optimized with random hill climbing. From the graph, we can see RHC quickly converging to the nearest optimum. Apart from the spikes, this is an entirely expected curve shape using a greedy algorithm. Because the neural net never hit its threshold, RHC continued to look for better neighbors until hitting the 1000 mark. A strange result from the data is that the testing error seemed to be lower than the training data. One possible reason for this is that the training and testing data were shuffled, meaning the distributions were roughly the same. It is also possible that there is uniform noise in the testing data that is not as prevalent in the training data. It may also be that the local optimum that RHC converged to happened to produce this result for the testing data. Doing cross validation would be a way to gain insight into the score discrepancies, i.e., whether this test was an isolated case or whether there is an underlying cause to this discrepancy. Because my experiment took over an hour to run and ABAGAIL does not directly support CV, I did not have time to implement, let alone rerun the tests with CV.

SA



Above are the training and testing curves for simulated annealing. Despite being heavily related to randomized hill climbing, SA produced a completely different curve. From the graph, we can infer that SA started near a local optimum. However, because my implementation of SA is designed to take more random walks near the start, we can see SA doing so around iteration 400, after which the error increases more than two-fold. This temporary increase in error is acceptable, because it is indicative of SA trying to escape its local optima. As shown in the graph, at around iteration 800, SA takes fewer random walks and converges to a local optimum. While not obvious in the graph, the error at iteration 1000 is 12% lower than the error at the beginning, which suggests that SA did indeed find a lower local optimum.

GA



Above are the curves for genetic algorithms. This graph looks like the randomized hill climbing graph, except that GA converged much more quickly. From the graph, we notice that the testing score is lower than the training score. The fact that this occurred twice suggests that the cause is some anomaly with the testing set we used, and not because of the peculiarity of the algorithms themselves. The optimization time for GA was over 50 minutes, yet it produced a worse score than RHC. One possible reason is because the problem structure was too complicated for GAs to capture. In that case, GAs tend to converge to local optima, as it seems to have done here. Nevertheless, every randomized optimization method except simulated annealing performed better than using back propagation. Had I more time, I would have experimented with different neural net structures, as well as different mutation and crossover functions for my GA. In addition, I would have implemented cross validation into my experiments in order to explain certain anomalies present in the data.

References:

De Bonet, JS., Isbell C, and Viola P (1997). MIMIC: Finding Optima by Estimating Probability Densities. Massachusetts Institute of Technology.