Why is Spambase interesting?

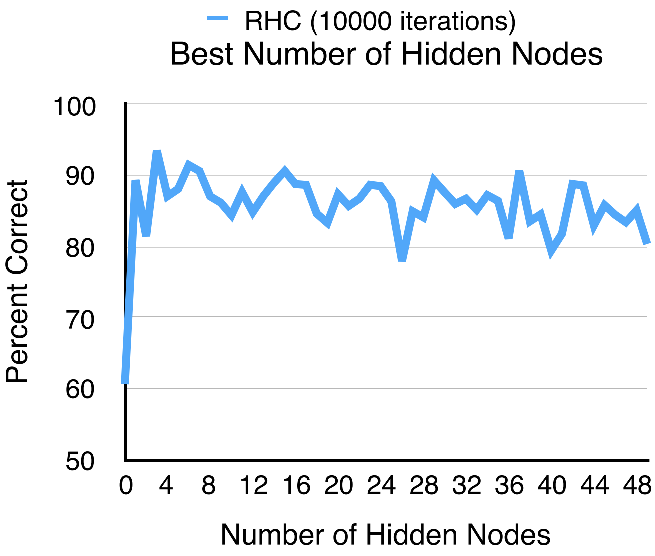
I chose to analyze a different dataset for project 2 as Abagail gave errors when dealing with nominal attributes. Spambase is a binary classification task; the classification decides whether or not an email is spam. There are 57 numeric attributes that describe the words in emails, and there are only 4600 examples, so this dataset is very sparse. The distribution of the values of the attributes is heavily skewed, as 0 is the most common value for most attributes, and it probably has a high kurtosis. Attributes include the frequency of various common words, the frequency of capital letters, the longest contiguous string of capital letters, and various information about punctuation. Spambase is an interesting dataset because email is an incredibly highly used medium for communication on the internet. Everyone has an email address, from corporations to individuals. Deciding whether or not an email is spam is a very difficult task and cannot be codified into one standard set of rules, as spam emailers change their techniques over time to attempt to avoid being caught by spam filters. Many companies currently solve this problem using machine learning techniques so that their spam filters can improve over time. Spambase is also interesting because through simple analysis of the word frequencies in the emails, algorithms can accurately predict (>90%) whether or not an email is spam.

Spambase Neural Network Analysis

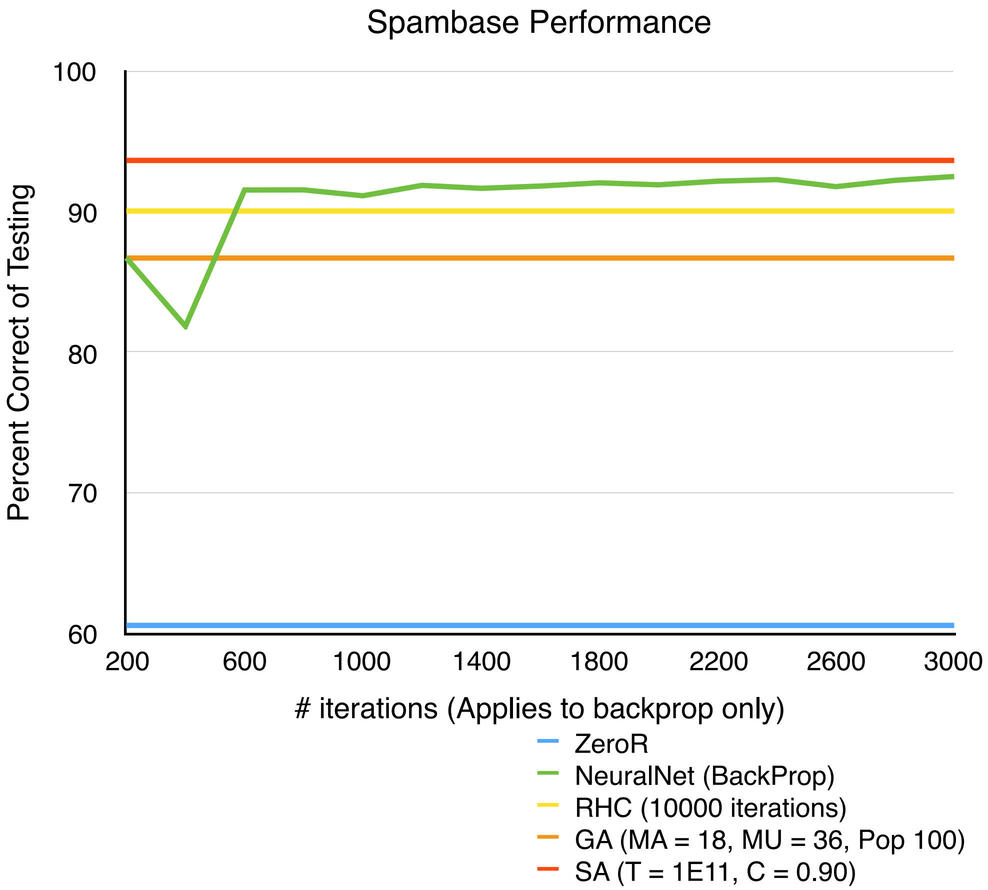
I split Spambase into a training set (70%) and a testing set (30%) by taking a uniformly random sample and then inverting that sample using Weka filters. I then tested various numbers of iterations to determine the accuracy of a multilayer perceptron neural network that uses back-propagation. At 600 iterations, the testing data was classified with 91.57% accuracy, and from that point it yo-yoed between 91% and 93%. The maximum was at 3000 iterations, at which the accuracy on the testing data was 92.53%. I used L=0.3 and M=0.2, the standard parameters. I compared this to classification by ZeroR, which is a baseline performance indicator that simply predicts the classification that has the majority in the dataset, and ZeroR had an accuracy of 60.6%. I suspect that 92.53% is very close to being the most accurate classification possible with this dataset without overfitting or adding more examples, as the researchers of the paper received a 7% error rate / 93% accuracy themselves. As this is a neural network, the hypothesis space consists of all of the real-valued weights for the connections between nodes.

Hidden Node & Iteration Hyperparameters for Random Optimization

I decided to use 10,000 iterations/restarts for each random optimization algorithm, as this was a time consuming operation and 10,000 iterations gave high levels of accuracy. I



assumed that the best number of hidden nodes for random hill climbing would be the same as for simulated annealing and genetic algorithms. I tested random hill climbing with 0-49 hidden nodes and found that there was some variability within the results. The classification accuracy stayed primarily between 80% and 90%, but after the local optima at the 5-15 hidden node threshold, the classification accuracy had on average a downward trend. I assumed that the average performance of random hill climbing at a number of hidden nodes would be similar to performance at a neighboring number of hidden nodes. With this assumption in mind, I used intervals of three data points and compared these intervals to one another and found that the interval with the highest average performance was at the point of 6 hidden nodes. Thus, I used 10,000 restarts and 6 hidden nodes for all of my later random optimization experiments.



Random Hill Climbing

Random hill climbing starts at a random point in the hypothesis space and then finds the closest local optima to that random point by “climbing the hill” to successively higher performing nearby hypotheses until it reaches a local optima. Random hill climbing randomly picked 10,000 points from the hypothesis space and found the nearest local optima to those points, and the final result was the best performing hypothesis found from one of the 10,000 restarts. RHC has flaws, as the search approach does not learn about the structure of the space from previous restarts as no data shared between each restart. Perhaps due to the simplicity of the algorithm, RHC is computationally inexpensive and runs very quickly. RHC was fast and surprisingly performant in finding neural network weights; its best performing hypothesis classified 90.074% of the examples correctly, performing only slightly less well than simulated annealing and back-propagation.

Simulated Annealing

Simulated annealing is similar to random hill climbing, but rather than taking a greedy approach and always finding the nearest local optima, SA at times randomly jumps to a new point in the space with a probability that is affected by the temperature (a measure of randomness) and the difference in classification accuracy between the two points. Thus, SA will explore more of the space and as the temperature decreases, will exploit the space like RHC.

For SA, I used 6 hidden nodes and 10,000 iterations. I tested out a variety of different temperature and cooling parameters, trying temperatures of (1E10, 1E11, 1E12, 1E13, 1E14, 1E15) and cooling rates at intervals of 0.05 from 0.55 to 0.95. On average, from all the various hyperparameters, simulated annealing classified 88% of examples correctly. On average, each temperature had almost the same performance, but had local optimums at different cooling parameters. I noticed that on average, at higher temperatures, the optimal cooling parameter would be lower. Lower cooling parameters cause the temperature to decrease at a higher rate, so it seems that there is a range at which random exploration of neighborhoods through high temperature is useful, and a threshold at which it becomes harmful to explore too much rather than exploit the space. The best performing point was at a temperature of 1E11 and a cooling rate of 0.90, and accurately classified 93.675% of the data. SA was the best performing out of all optimization algorithms and was computationally inexpensive, but not as much so as RHC.

Genetic Algorithms

Genetic algorithms work by training a random set of learners called the population in parallel, and at each iteration, mutating some of them to locally exploit the space, and mating the higher performing ones to generate a new hypothesis that is the combination. Mutation is similar to random hill climbing, but mating allows one to share significant amounts of data between learners in the population and learn about the space one is working within. I tested GA with 6 hidden nodes and 10,000 iterations. I tested out a variety of different population sizes (100-400 at increments of 100), as well as mutation and mating numbers (9-36 at increments of 3). On average, as the population size increased, the accuracy decreased; this may be because I didn't increase the number of iterations with the population, and larger populations may take longer to converge. In general, the higher the amount of mating between members of the population, the higher the classification performance of the best hypothesis. Genetic algorithms were slightly lower performing than the other algorithms, with only 87% classification accuracy.

Traveling Salesman:

The traveling salesman problem is a graph search problem in which one attempts to find a traversal of a graph with N nodes that travels through each node once and returns to the starting node with the lowest distance traveled. I ran my randomized optimization algorithms against an instance of the traveling salesman problem with 25 nodes. I used a variety of hyperparameters per algorithm, and ran 50 trials with each set of hyperparameters. I then took the average and best result from each combination of hyperparameters.

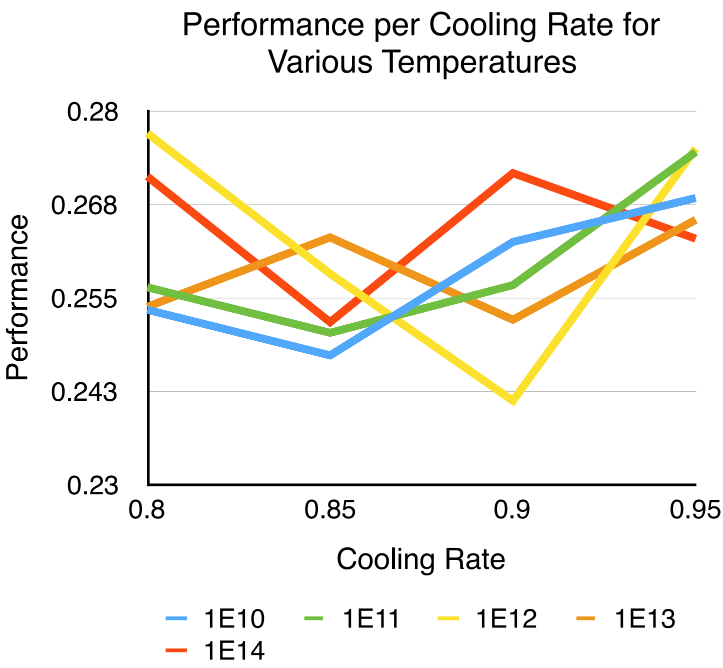
	RHC	SA (T=1E12, C=0.8)	GA (MA=60, MU=3)	MIMIC (Samples=250, toKeep=20)
Average	0.2182	0.2214	0.2652	0.4171
Best	0.2766	0.2771	0.2774	0.6306

Random Hill Climbing

I ran 50 trials of RHC. Each trial had 20,000 iterations. This finished in an order of magnitude less time than genetic algorithms took to run, but the best result found (0.2766) was very close to the best hypothesis found by genetic algorithms. RHC had the lowest average performance (0.2182) out of the algorithms. This is expected as its greedy approach only explores a tiny portion of the space, and the algorithm does not share any information between its instances to learn about the structure of the space.

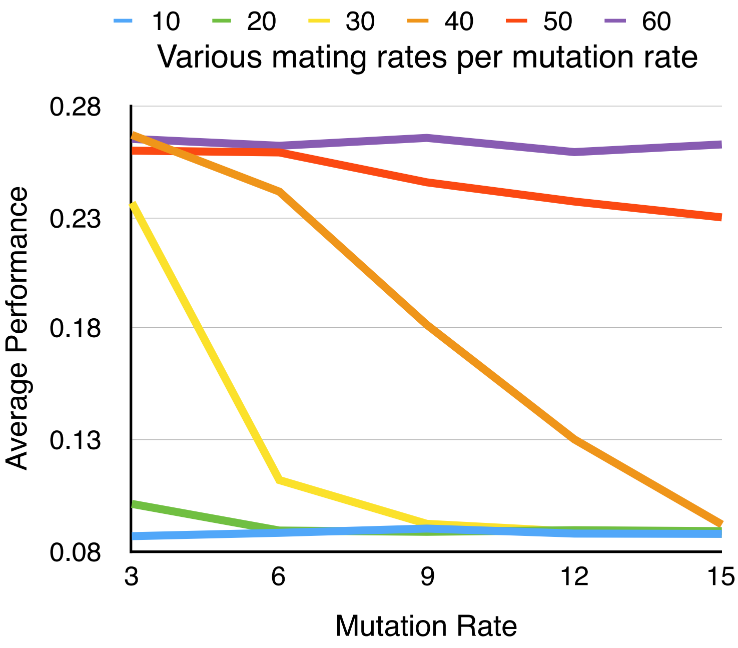
Simulated Annealing

I tested temperatures 1E10, 1E11, 1E12, 1E13, and 1E14; I tested cooling rates from 0.8 to 0.95 with intervals of 0.05. I ran 50 trials per pair of hyperparameters. The best average result and best result in general was with a temperature of 1E12 and a cooling rate of 0.8. It seemed that there was a local optima at temperatures of 1E11 and 1E12, but there was no discernible relationship between changes in cooling. SA performed a little better than RHC, but its average results (0.2214) and best results (0.2771) were only marginally better. This makes sense. Though SA explores the search space more than RHC, it only retains a single best point from that exploration and doesn't learn much about the structure of the space.



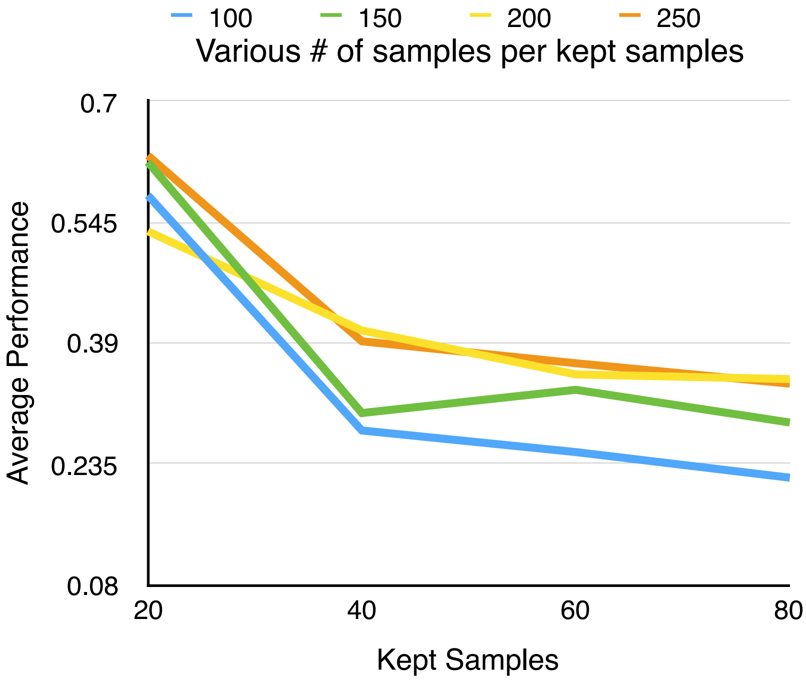
Genetic Algorithms

I ran GA with a population of 200 and 1000 iterations per trial. I ran 50 trials per different pair of hyperparameters of mutation and mating. I tried the mating hyperparameter at intervals of 10 from 10-60 and the mutation hyperparameter at intervals of 3 from 3-15. Lower numbers of mutations consistently gave better results, and higher amounts of mating increased the results. The best result from GA (0.2774) was only slightly higher than that of SA, but I think if I had increased the number of iterations per trial this would have improved significantly. GA took quite a long time to run. The average result (0.2652) from GA was much higher than that of SA and RHC.



MIMIC

I ran 50 trials of MIMIC per pair of hyperparameters with 1000 iterations per trial. MIMIC is an algorithm that estimates the location of local maxima within a hypothesis space by attempting to approximate the distribution of high performing hypotheses in the space, sampling from that distribution, and improving that distribution by only keeping some number of the best samples at each iteration, then basing the new distribution off of those kept samples. MIMIC was by far the best performing algorithm for this problem, with an average result of 0.41 and a best result of 0.63, more than double the next highest performing algorithm, GA. MIMIC performed better with higher numbers of samples and keeping a low proportion of those samples. I performed 50 trials with 1000 iterations per trial, and varied the number of samples from 100 to 250 by increments of 50, and the number of kept samples from 20 to 80 at increments of 20. MIMIC took a really long time to run per iteration, as it had to approximate the distribution each time; it took perhaps hundreds of times longer to run than the next slowest algorithm.



Four Peaks:

Four peaks is a classic function approximation problem in which one gives as input the size of a bit string (N) and a trigger point T, an integer (0 < T < N). Four peaks is at its global maximum when it has all 0s or 1s up until T, then all of the opposite bit from that point onwards. Four peaks is interesting because it always has four local maxima. I used a variety of hyperparameters per algorithm, and ran 50 trials with each set of hyperparameters. I then took the average result from each combination of hyperparameters. I ran Four Peaks with an input size of 100 and a trigger point at index 25.

Percent of Results with Suboptimal Value (lower is better)	RHC	SA	GA (MA=10,MU=9)	MIMIC (Samples=100, toKeep=80)
Average	100	100	1.42	6.14

Random Hill Climbing

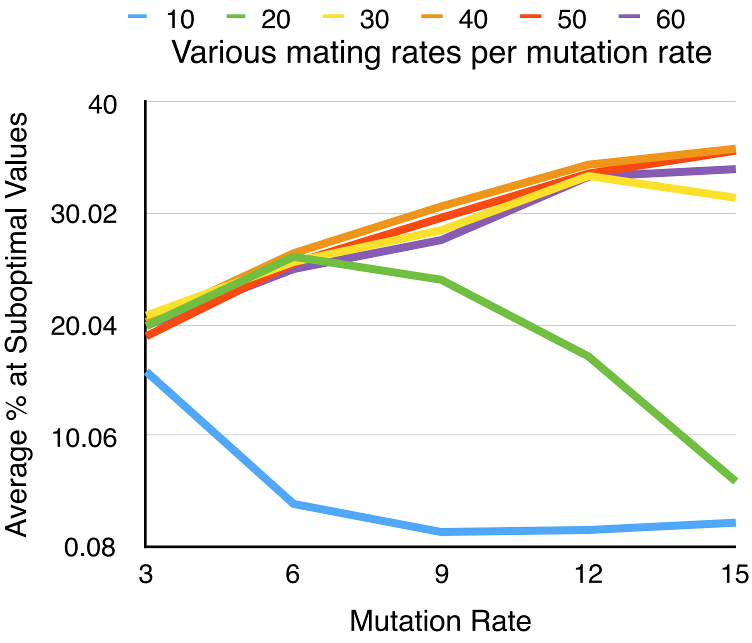
I ran 50 trials of RHC. Each trial had 20,000 iterations. I ran it with 20,000 iterations per trial. Random hill climbing always returned a suboptimal value. This is typical of a greedy approach, as RHC does very little exploring of the space and will always settle into the nearest maxima, which can often be a local maxima. 100% of the trials from RHC received suboptimal values. This makes sense, as not only is RHC greedy, it also does not share information about the space between instances. So each instance of RHC stumbles around blindly (randomly) to the first local maxima.

Simulated Annealing

I ran 50 trials of SA per pair of hyperparameters. I ran it with 20,000 iterations per trial. I tried SA with temperatures of 1E10 through 1E14 incrementing by powers of 10, and cooling rates of 0.80 through 0.95 at increments of 0.05. Surprisingly, despite the fact that SA explores the space much more than RHC does, it did not find the global maxima even once. 100% of the trials for SA also received suboptimal values. This is a bit surprising, but the search space is quite large and the four local optima take up a majority of the space.

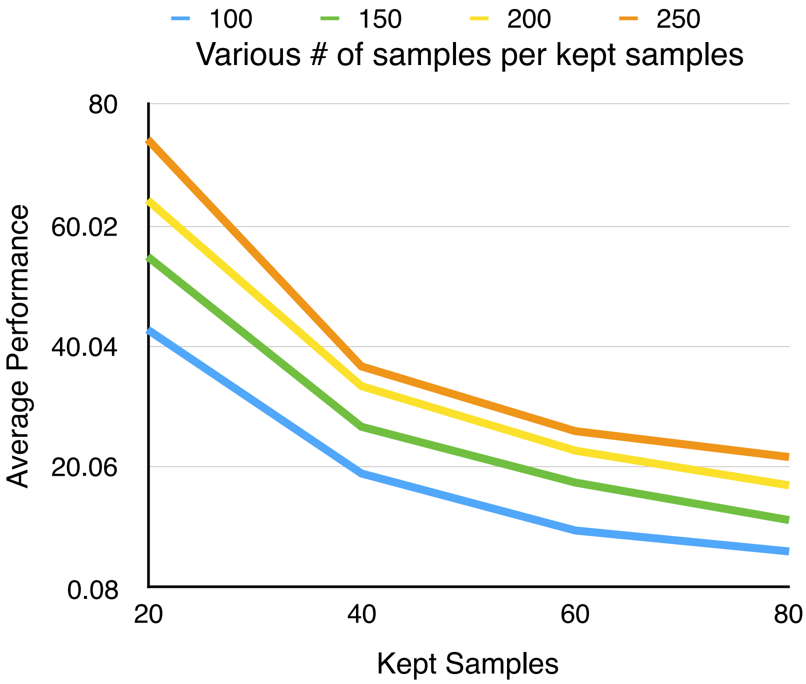
Genetic Algorithms

I ran 50 trials of GA per pair of hyperparameters. I ran it with a population of 200 and 1000 iterations per trial. GA performed ridiculously well on this problem. At the best mating and mutation values, GA only missed the global maximum and got to a suboptimal value 1.42% of the time. GA obviously learned the structure of the space very well by crossing over pieces of information between learners that were performing better; perhaps some learners were choosing 0s up to the trigger point and thereby increasing their score, and others were choosing 1s past the trigger point and then they mated and got even higher results. GA took about ten times longer than SA to run, but the results were more than worth it. The higher the mating rate, the worse the results were, but the relationship between mutation rate and results varied. For the optimal result, there was a local optimum for the mutation rate at 9, before and after which results worsened. Also, I think GA had such a high rate of optimality because all of the optimal learners mated multiple times, wiping out almost the whole rest of the population.



MIMIC

I ran 50 trials of MIMIC per pair of hyperparameters. I ran it with 1000 iterations per trial. MIMIC was the second highest performer on this problem, far outstripping SA and RHC, but not quite getting the incredible performance of GA. MIMIC missed the global maximum and got to a suboptimal value 6.14% of the time. It was able to calculate the distribution of high-performing hypotheses very well. Note that Four Peaks has better performance for lower values on the graph. As the number of kept samples increased, the accuracy improved. As the number of samples overall increased, the accuracy decreased. MIMIC did well on this problem as it directly estimates the location



of local maxima, and improves its distribution of where they could be over time, and that is almost perfect for this type of problem.

Flip Flop:

Flip flop is a problem that counts the number of “flips” between bits in a bitstring. It counts the total number of contiguous bits that are of different values. This function has a ton of local optima, as at any given point there could only be a few bits that could be flipped to improve the performance where flipping most of the bits would decrease the performance. I tested against Flip Flop with an input size of 150, so a bit string of length 150.

	RHC	SA (T=1E14,C=0.8)	GA (MA=60,MU=15)	MIMIC (Samples=150, toKeep=20)
Average	121.28	145.66	124.64	135.8

Random Hill Climbing

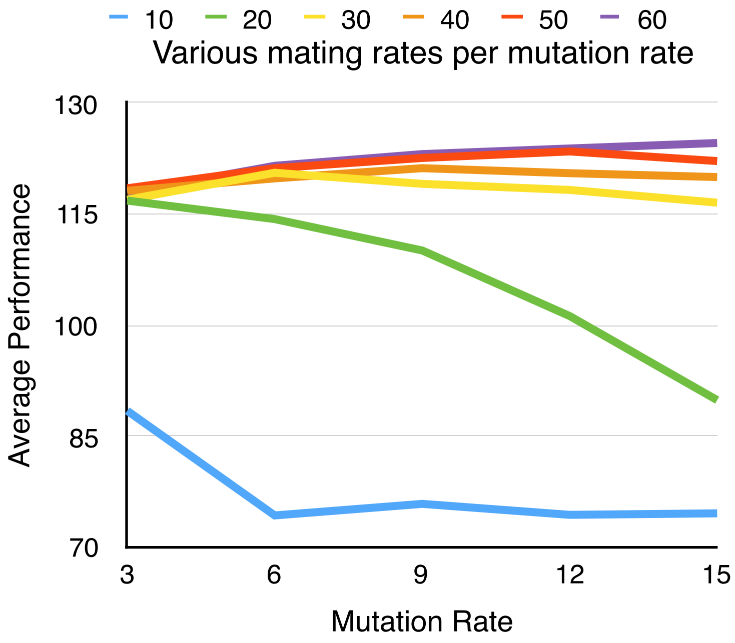
I ran 50 trials of RHC. Each trial had 20,000 iterations. I ran it with 20,000 iterations per trial. RHC performed the worst out of all of the algorithms. There are a lot of local maxima in Flip Flop, so RHC is unlikely to find the single correct permutation just by greedily modifying its hypothesis function repeatedly. RHC will almost always get stuck in one of these local maxima.

Simulated Annealing

I ran 50 trials of SA per pair of hyperparameters. I ran it with 20,000 iterations per trial. SA functioned incredibly well for this task, exploring the space with depth. Interestingly enough, SA performed almost equally well with every single parameter pairing. I tested temperatures 1E10, 1E11, 1E12, 1E13, and 1E14; I tested cooling rates from 0.8 to 0.95 with intervals of 0.05. Every pairing had a performance of over 140. This is an ideal problem for SA, as it will keep jumping around the search space while its temperature is high, and slowly hone in on a single maximal solution. This allowed SA to explore a ton of configurations and avoid getting stuck in most local maxima.

Genetic Algorithms

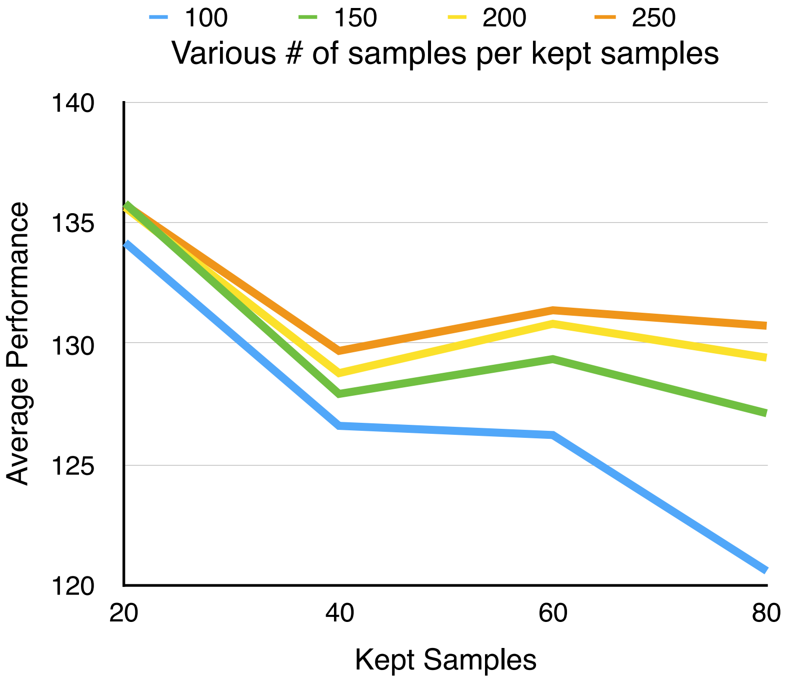
I ran 50 trials of GA per pair of hyperparameters. I ran it with a population of 200 and 1000 iterations per trial. When there were lower mating rates, increasing the mutation rate



caused lower performance, but once the mating rate was 30 or higher, increasing mutation started to increase performance up to locally optimal point. The best result occurred with the highest mating rate of 60 and the highest mutation rate of 15, and the performance was 124.64, barely better than RHC. This algorithm grossly underperformed SA and MIMIC. This was most likely because of a low population size and a low number of iterations. This problem is very well-suited for being improved by crossover, as a contiguous string of bits that are flipped back and forth could be crossed over to a learner with a contiguous string of flipping bits at another section of the string for improved performance, so I'm surprised at this result.

MIMIC

I performed 50 trials with 1000 iterations per trial, and varied the number of samples from 100 to 250 by increments of 50, and the number of kept samples from 20 to 80 at increments of 20. MIMIC was the second highest performer, and approximated the distribution of local maxima very well (score of 135.8). There were so many local maxima that I think MIMIC would often sample out a lower local maxima just by chance. We see a general downward trend in accuracy for MIMIC with an increased number of samples kept.


Various # of samples per kept samples
Average Performance
Kept Samples