Assignment 2:
CS 7637

Ryan Johnson
ryan.johnson@gatech.edu

Abstract- Four randomized optimization algorithms were implemented across three different optimization problems as well as the task of optimizing a neural network. We looked at the trade off in fitness, execution time, and hyperparameters across all algorithms. Genetic algorithms seem to perform best in this trade off for the Four Peaks problem and the Neural Network weights optimization. MIMIC performs strongest in the Knapsack optimization problem, and Simulated Annealing's strengths can best be seen in the Flip Flop optimization problem.

I. OPTIMIZATION PROBLEMS

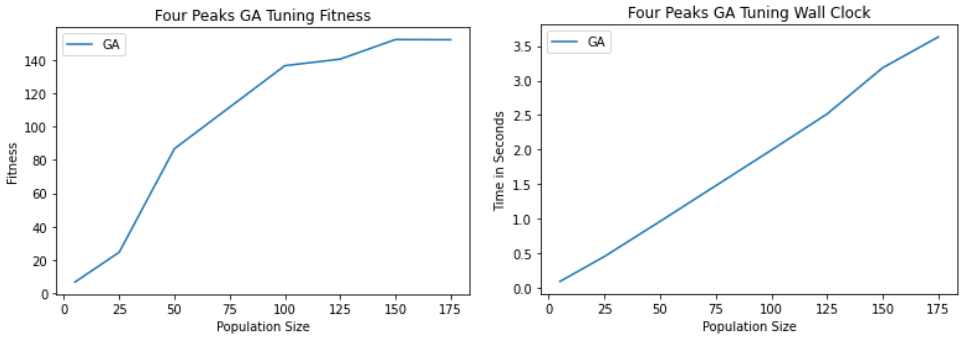
1 METHODOLOGY

For 3 discrete maximization optimization problems, we want to analyze the different performance of the randomized optimization algorithms: Randomized Hill Climbing, Simulated Annealing, Genetic Algorithms, and MIMIC as implemented in MLrose. In our experiment, we will look at algorithm fitness per iteration and overall wall clock performance. Additionally, we will test each algorithm as the problem space grows in terms of both fitness and wall clock performance. Because these are randomized optimizers, for each problem we average together the runs of multiple optimizations. This will give us a sense for algorithm performance as a whole rather than happening upon one lucky run.

Hyperparameters for each model were tuned to find values that work generally well across all optimization problems. It's certainly possible that with further tuning some algorithms could be improved on some problems, however, for the sake of comparison and conciseness in the number of charts displayed, the algorithms were tuned to use the following hyperparameters:

- Randomized Hill Climbing: restarts = [0,3,6], iterations = 500
- Simulated Annealing: temperature = [.01,1,1.5,3,5], decay = [expo, geom], iters = 500
- Genetic Algorithms: population = [25, 50, 100, 150], mutation_rate = [1%, 10%, 20%], iters = 500
- MIMIC: sample_size = [25, 50, 100], keep_percent: [10%, 20%, 50%], iters= 500

Tuning hyperparameters is an important element of each of these problems. Below we see the hyperparameter tuning done for GA across various population sizes. As the population grows, the search net of the genetic algorithm grows broader as there are more chances for an individual to be close to one of the peaks, however, we see our first example of the fitness/time trade off which in many ways is similar to the bias variance trade off we saw in supervised learning.



As our population increases, fitness initially increases very rapidly before somewhat tapering off. However, the wall clock speed of the algorithm increases linearly. There seems to be a point that balances fitness with wall clock speed at a population size of about 100. Thus, we select 100 as our population, though, if we were unconcerned with time performance, we may select 175 as our population. This tuning and trade off was done for each algorithm with the hyperparameters listed above. In some cases, these made a large difference like we've shown above. We've highlighted some differences throughout this paper, but also focus on the core underlying differences between the algorithms.

2 PROBLEM DESCRIPTION AND HYPOTHESES

2.1 Knapsack Problem

The knapsack problem is fairly straightforward, but quite difficult and computationally intensive. Essentially, for some set of items with an associated weight and value, what combination of items should be fit into our weight limited "knapsack" maximizes the value contained represented by:

2.2 Knapsack Hypothesis

Knapsack is a combinatorial optimization problem that is not NP-complete, meaning there's no known polynomial algorithm that can tell whether our solution is optimal. With this understanding, my hypothesis is that either Genetic Algorithms or MIMIC will perform best here. It stands to reason that there are many local optima with different possibilities and I imagine Simulated Annealing and Hill Climbing could be prone to get stuck within them while GA and MIMIC are better able to sample across the search space.

2.3 Four Peaks Problem

Four Peaks is an intuitive optimization problem given by (Baluja and Caruana, 1995). Effectively, given a bitstring vector of some n length, we create 2 global maxima and 2 local maxima. However, further

increasing the difficulty the local optima have wide basins throughout the middle of the search space with the global maxima residing at the extremes.

2.4 Four Peaks Hypothesis

Four Peaks seems designed to be a pitfall for both RHC and SA. Narrow maxima and wide gradients ascending to the local maxima should be very effective at confounding our hill climbers. I would expect Simulated Annealing to perform better than RHC, but I doubt SA's exploration vs exploitation will be enough to overcome the basins. However, GA and MIMIC should perform quite well given their ability to explore and sample the population. Furthermore, in Baluja and Caruana's work, they identify the mechanism for GA's likely finding of the narrow global basins and their ability to then crossover that relatively high fitness (1995). For this reason I would expect GA to perform best

2.5 FlipFlop Problem

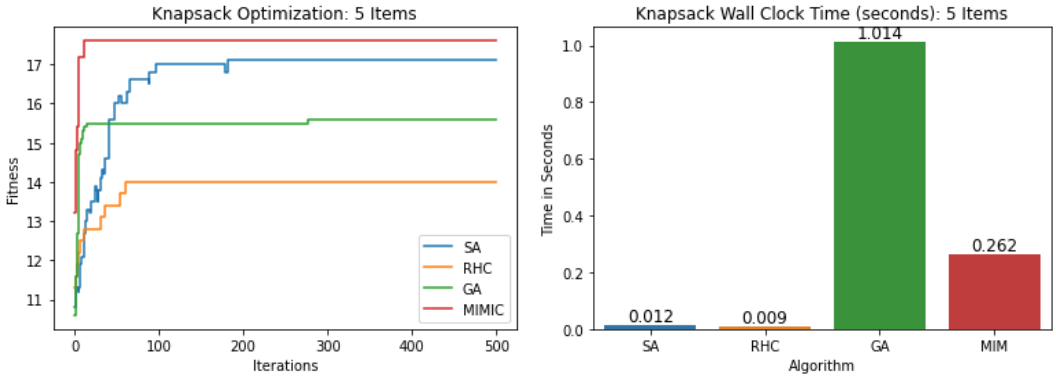
The flip flop problem is deceiving in its simplicity. The problem simply counts the number of "flips" in a bitstring between 0 and one. So a bitstring with only a single value will have a fitness of 0 while a bitstring with a "01010101" alternation will have the maximum fitness.

2.6 FlipFlop Hypothesis

There are no obvious traps in the flip flop problem like we see in four peaks, nor do we expect as many local minima as seen in knapsack. For this reason, I expect all algorithms to perform well and the best algorithm will likely come down to that trade off between fitness and runtime. Given this, simulated annealing seems like the most likely to perform well.

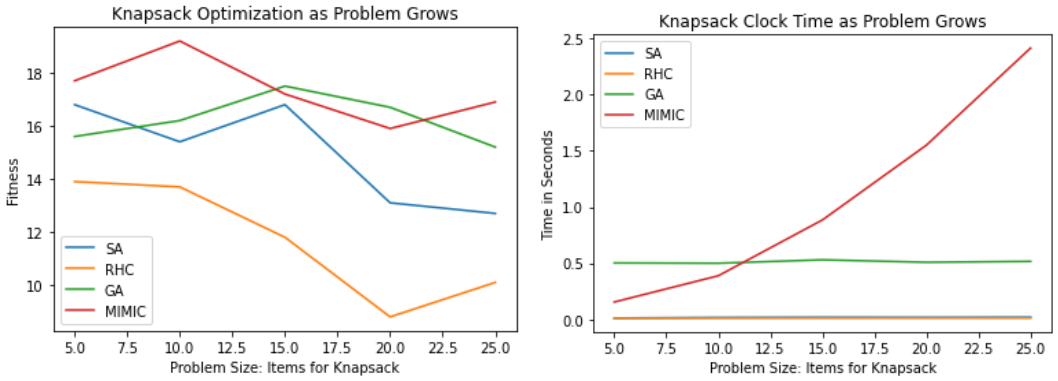
3 EXPERIMENT RESULTS

3.1 Knapsack Results



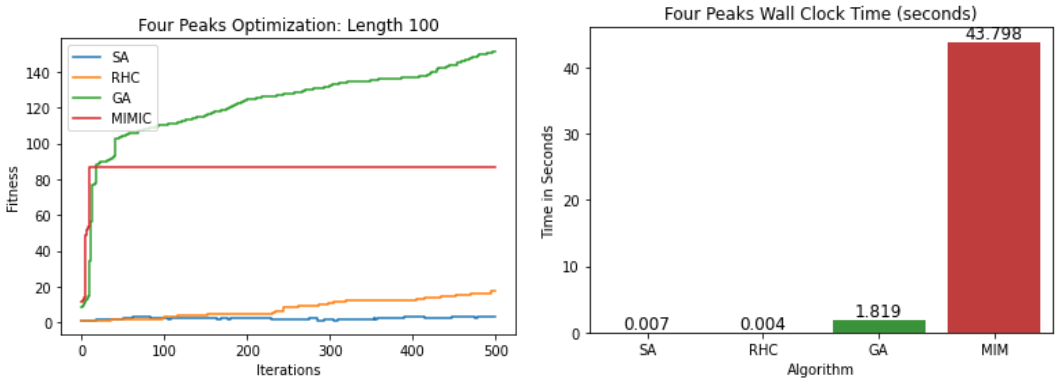
For knapsack optimization across 5 items with varying weights and values, we see decent performance across all algorithms except for random hill climbing which falls behind. This result shows mimic

outperforming the other algorithms. This is likely due to its ability to represent the structure even in an NP-hard problem. MIMIC does particularly well here in reducing the number of iterations to reach a fitness plateau, and additionally solves the problem more quickly than some of the other optimizations we will see in the other problems. In this problem it's beneficial to try to match the underlying structure for the problem as there can be unclear improvements and relationships from item to item when using a greedy hill climber..



As the number of items in our knapsack grows, we see performance slowly decaying across all algorithms. MIMIC and GA seemed to be the most resistant as their ability to represent structure fits the otherwise loosely organized fitness function. As for clock time with problem growth, we see MIMIC outperforms GA up until about 10 items where it begins to take a bit longer. This is representative of MIMIC's exponential execution time, but it does quite well here when the space is slightly smaller and the fitness vs clock time balance can look more favorably on MIMIC. As the problem space grows larger, it's likely GAs advantage would start.

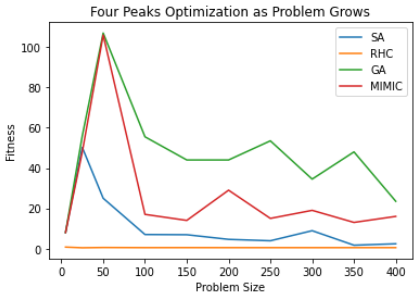
3.2 Four Peaks Experiment Results

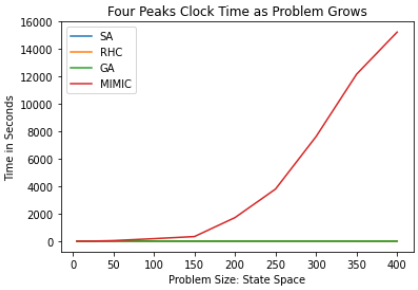


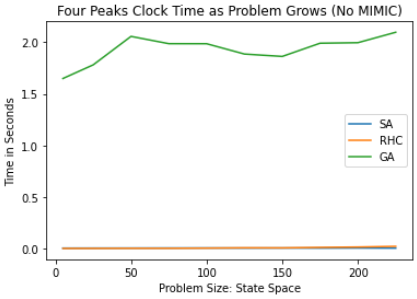
For four peaks optimization, we ran the problem a dimension of 100 and averaged together the average fitness iteration across multiple runs. GA and MIMIC were both found to perform well with larger

populations and sample sizes we used 150 and 100 respectively. We can see that the genetic algorithm is far and away the best performer, nearly matching MIMIC's performance in nearly as many iterations. Then, it continues to improve throughout the remaining iterations. SA and RHC, as expected as greedy hill climbers, both get caught in the local minima and perform very poorly even with respect to their impressively quick wall clock run times. The ability for SA to search is meaningless when it first defaults to always climbing the hill when possible. The temperature is just not high enough to continue to search beyond the wide troughs along a local minima.

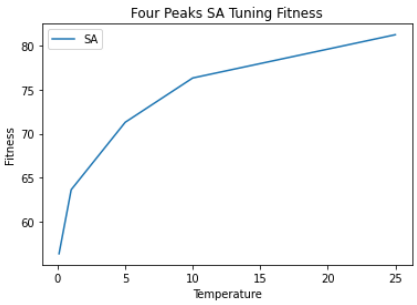
Meanwhile, the genetic algorithms make very little trade off between runtime and fitness. We see a consistent runtime of about 2 seconds and easily the best fitness. The genetic algorithm is able to do this because it's likely one of the individuals in its population will simply happen to reside near a global maxima. This individual then refocuses the development of each iteration towards that maxima via crossover. This results in the rapid gain in fitness right at the start and then the continued gradual improvement as each generation approaches the maxima.





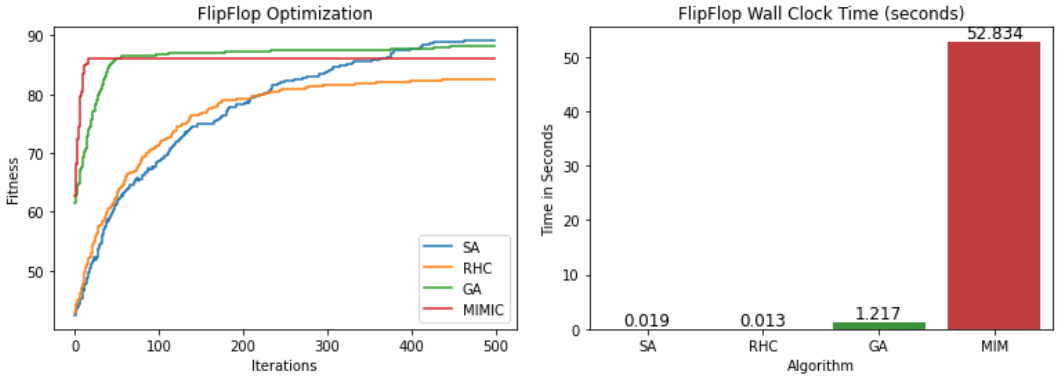


As the problem grows large, we see the GA maintains this advantage through some very massive optimizations. The genetic algorithm is only really challenged by MIMIC at smaller problem sizes where mimic's sampling is effective at representing an overall structure. While the problem becomes harder as the search space increases, the GA maintains respectable performance as MIMIC's performance balloons towards truly impressively long runtimes.

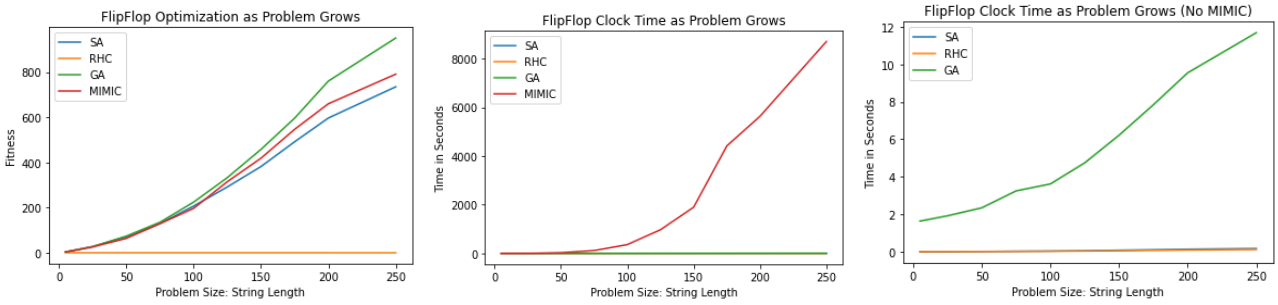


Notably with four peaks, we can also greatly improve the performance of the simulated annealing algorithm by increasing temperature. This allows the algorithm more time to walk into the narrow bands of the global maxima, though, in order to achieve this, we needed to drastically increase the number of iterations SA ran thus this isn't entirely reflected on the learning curve above. Even still, we come short of GA's performance. With this in mind, GA can be selected as the strongest optimization algorithm for four peaks.

3.3 FlipFlop Experiment Results



Our experiment tested each algorithm across a bitstring of length 100. SA preferred a lower temperature than four peaks using 1. GA had found a performance balance with a population of 100 as did MIMIC's sample size. Across all 4 algorithms we see respectable performance. This is likely due to a lack of clear local minima for the hill climbers to become stuck in. MIMIC was able to converge in the fewest number of iterations followed closely by the genetic algorithms, however they were both eventually surpassed by the simulated annealing algorithm. Further, the wall clock execution time of SA was 10x faster than the genetic algorithm and several orders of magnitude superior to MIMIC. While MIMIC and GA have to keep hypotheses in memory, via the samples for MIMIC and the population for GA, SA is able to rapidly explore and exploit.



As the problem scales, we see SA continue to perform strongly, though, in some instances it is outdone by GA and MIMIC in the largest string lengths. This is somewhat accountable to the underlying randomness of the algorithms, but also represents GA's strong ability to search a broad search space. However, while slightly outperforming SA in fitness, when we look at the trade off between the algorithm's clock times, we see that SA has scaled very nicely maintaining its sub .1 second run time. While all algorithms were run with the same number of iterations for this experiment, given the massive difference in execution time, we could feasibly run 10x the number of SA iterations in the time it takes to execute GA, likely improving even further on simulated annealing's fitness.

II. NEURAL NETWORK

1 METHODOLOGY

Using the same Breast Cancer dataset from the UCI from assignment 1, we trained a neural network on scaled data to eliminate any bias in the size of input features. The neural networks were configured with the exact same hidden layer node structure ([10, 10, 10]) and activation function (relu) across 4 different methods for optimizing the weights and bias of the network structure: backpropagation and three randomized optimization algorithms randomized hill climbing, simulated annealing, and genetics algorithms. Backpropagation was implemented using the mlpClassifier in scikit-learn and the randomized optimization algorithms were implemented using MLrose. Each algorithm's hyperparameters were tuned accordingly in order to maximize k-fold test accuracy. The learning curves, wall clock, and tuned hyperparameters are then compared to backpropagation in order to analyze and understand the differences in the algorithm.

2 HYPOTHESIS

One of the big problems facing optimization of neural network weights and bias vs the optimization done in part 1 of this paper is the nature of optimizing a continuous space of variables rather than the discrete optimization seen above. A continuous, real-valued optimization space means these algorithms could tweak and move weights around forever in an infinitely large search space.

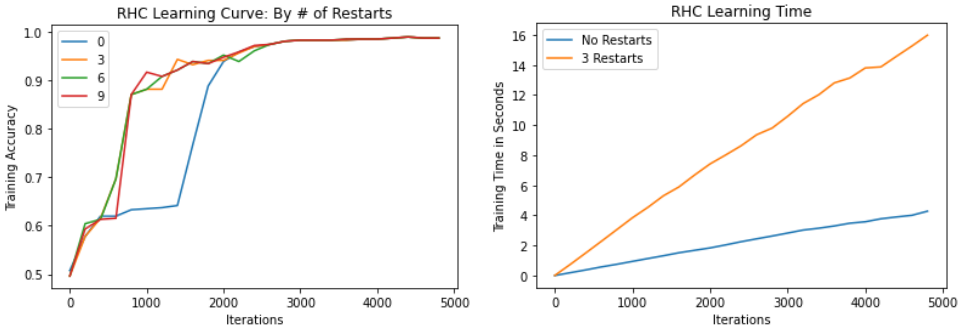
Backpropagation is a proven and efficient solver that uses gradient descent to optimize the weights of a neural network. It is quite well suited for the problem at hand. I would expect to see a more rapid learning curve as this algorithm is much more targeted than the random optimizers. However, it is still capable of getting stuck in local optima, so it's possible one of the randomized optimizations will find a global maxima and perform better given enough time and iteration. However, given the inherent challenges present with continuous optimization, I expect backpropagation to perform best across any set of time, performance, iteration, and bias variances trade offs.

3 RESULTS

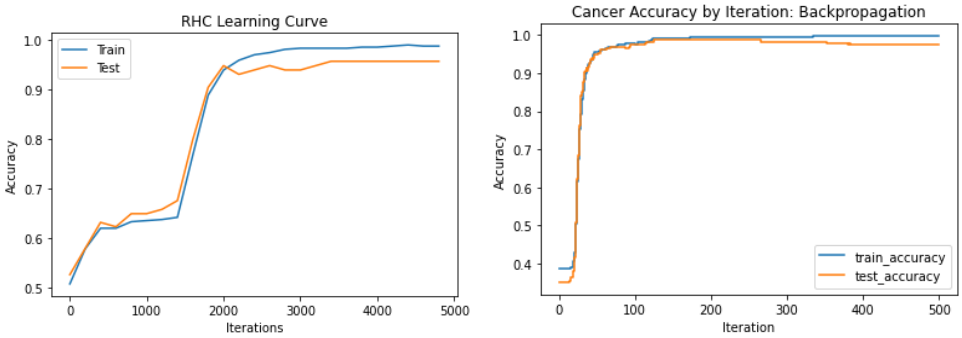
Below are the results of each algorithm after hyperparameter tuning. In line with our hypothesis, backpropagation performed best across all measurements. However, I was still impressed with very strong performances from each of the other algorithms which will be explored in detail individually. If training time is a concern, outside of backpropagation, RHC performs best in terms of the fitness/time trade off, though it does have a quite large validation variance. If training time in no concern, GA performed best with strong bias, minimal variance, and few iterations at the cost of wall clock time. With this in mind, however, SGD clearly wins overall.

Algorithm	Training Acc	Validation Acc	Wall Clock Time	Iterations
Backpropagation	0.9912	0.9868	0.08 seconds	150
RHC	0.9824	0.9561	3.937 seconds	3400
SA	0.9714	0.9474	9.556 seconds	8500
GA	0.9802	0.9737	25.492 seconds	375

3.1 RHC Experiment Results



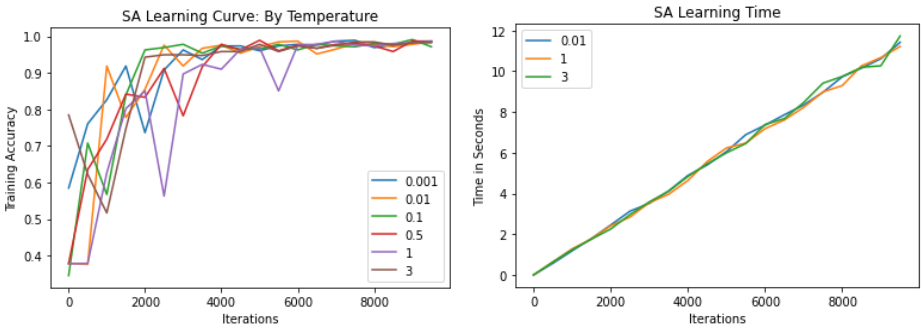
Our RHC algorithm was tuned by iteration as well as the number of restarts. By restarting the algorithm and attempting a random search a second time after reaching either a convergence optima (which is unlikely in a continuous optimization problem) or reaching the end of the iterations. This gives the algorithm additional attempts to avoid getting stuck in a local maxima and instead optimize from a different part of the function. We can see above that these extra restarts did help our algorithm in the earliest iterations, but in the end the RHC found the same maxima regardless. Because running restarts effectively reruns the optimization, we see effectively a wall clock time scaled by the number of restarts. For this reason we used zero restarts in the final optimization.



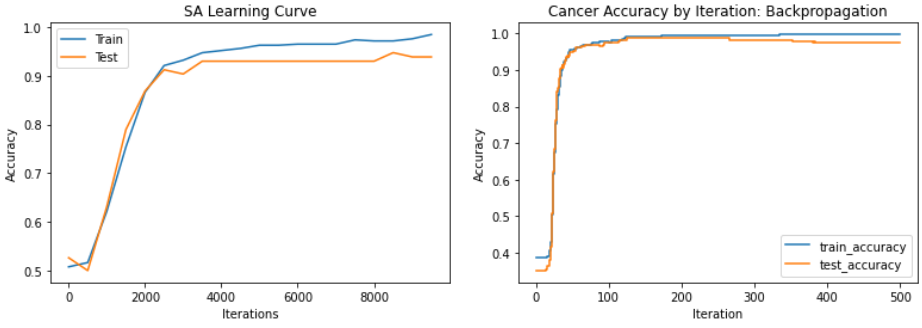
Randomized Hill Climbing performs closest to Stochastic Gradient Descent by execution time, and isn't far off training accuracy. However, we do see a strong bias variance trade off in the RHC algorithm in

the form of a slightly stronger over fit than the SGD. Note the scale of the X axis is quite different between the charts, as well. RHC took many more iterations to converge close to the convergence of SGD. This is simply a question of efficiency as RHC slowly changes one weight or bias at a time vs SGD's backpropagation. This being said, I'm not surprised that RHC does seem to perform similarly to backpropagation. After all, hill climbing is very similar in principle to gradient descent.

3.2 SA Experiment Results



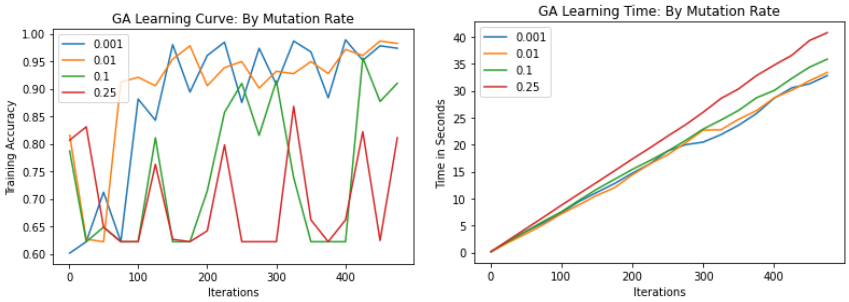
Simulated Annealing's hyperparameters were tuned specifically around its starting temperature hyperparameter. This temperature has the ability to change the behavior of the SA algorithm between behaving more like a random walk vs more like random hill climbing. I hypothesized a great temperature value may improve SA's performance against something like RHC, but this was not the case. In order to account for this additional search, we executed twice the number of iterations within simulated annealing compared to random hill climbing. Across different starting temperatures we see highly variable performance in the early iterations. This makes sense as the algorithm may walk into less optimal states in order to improve in the long run. As the number of iterations increases, however, we see all temperatures converge to similar values. This tells us we are behaving more like RHC as temperature cools, but also that no walks were able to escape a local optima if there is one at all. With this in mind, we select .01 as our temperature value.



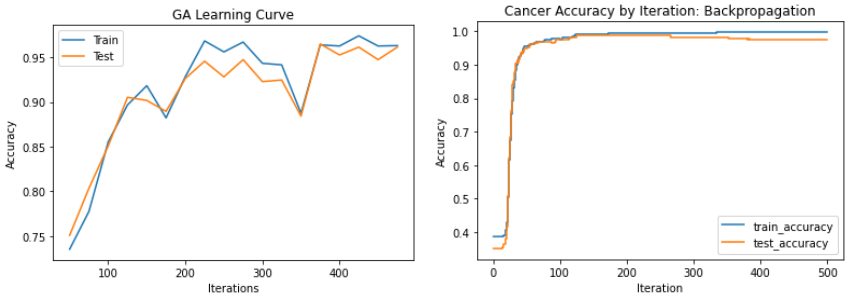
Simulated Annealing performed worst among the tested algorithms. Taking the longest to converge in terms of iterations, overfitting the most with large variance, and performing the worst in terms of

accuracy. In this case, a random walk was not needed in order to optimize. Given more iterations, it's possible SA could improve and could likely match RHC, but for what purpose when RHC performs better.

3.3 GA Experiment Results



Mutation rate had the biggest impact on the genetic algorithm's performance. As seen in the above chart, a larger mutation percent has wildly erratic training accuracy from iteration to iteration. This is caused by the algorithm effectively mutating itself out of an optima. The weights of a neural network are quite sensitive and too many mutations can get in the way of consistent progress being made. On the other hand, the smaller mutation rates were able to maintain the general trend of converging fitness while still searching to find improvements. This comes with the benefit of being slightly faster, as well, as the algorithm doesn't have to apply as many mutations to the individuals in the dataset. For this reason we select .01 as the mutation rate.



While the path towards convergence is the least smooth of all algorithms with the genetic algorithm, we see some of the best performance. GA sees the least overfitting in the best tuned model with the greatest training accuracy outside of SGD. It's also the only algorithm to converge in a number of iterations somewhat close to SGD. While it's computationally intensive, the population of the GA is able to search a wider area in fewer iterations than SA or RHC. However, maintaining this sort of memory and executing crossover each generation comes with the side effect of being the worst in terms of wall clock time.

REFERENCES

Baluja, & Caruana. (1995, May 1). Removing the Genetics from the Standard Genetic Algorithm. ICML.

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

Bertsimas D & Tsitsiklis (1993). Simulated Annealing. Statistical Science