RANDOMIZED OPTIMIZATION ANALYSIS

Shannon Ke
CS 4641
Randomized Optimization Report

Local Random Search Algorithms

Dataset Description

Using the cars dataset from the previous project, I constructed a neural network that was optimized using three strategies: randomized hill climbing, simulated annealing, and a genetic algorithm. The cars dataset consists of discrete data values, with six attributes, 1728 total examples, and four class labels. The dataset focuses on evaluating cars based on different attributes and determining whether or not the car is acceptable or unacceptable, making it a very practical problem to build machine learning models off of.

General Methodology

Using a machine learning library already implemented in Java called ABAGAIL, I was able to build and test the optimization strategies. After encoding the labels in the cars dataset csv file using a python script I named data_util.py, I plugged it into my CarsTest.java file included in the opt.test package in ABAGAIL. For all of the following operations, I kept my data consistently at a 70-30 split, with the former being the percentage kept as training data and the latter kept as testing data. Cross validation involved k-fold validation with k equaling 10, although I did not analyze validation values much.

I ran all three optimization strategies on a neural network built from the cars dataset, keeping hyperparameters for the neural net at their default values. During the first run, I played around with the hyperparameters for each algorithm to determine optimal values for the second run, with randomized hill climbing being the exception. Since randomized hill climbing does not take in hyperparameter values, the only parameter that was altered to produce data was number of iterations. However, for simulated annealing, I played with both temperature and cooling rates, and for the genetic algorithm, I altered mating and mutation ratios. Thus, I was able to determine what the optimal hyperparameters were based off of the produced graphs, and did second run through of my data, keeping the hyperparameters constant and changing only iteration number so that I could produce data that could adequately compare all three optimization problems in a more general sense.

Randomized Hill Climbing

With randomized hill climbing being a relatively simple optimization problem, given that it does not have tunable hyperparameters save number of random restarts, it acts as a good basis of comparison for the other algorithms. In order to get a sense as to how random hill climbing converges, I ran it on the cars dataset with varying numbers of iterations. I do this later as well,

with the two other optimization problems represented as well. See *Figure 4* and *Figure 5* for those comparisons.

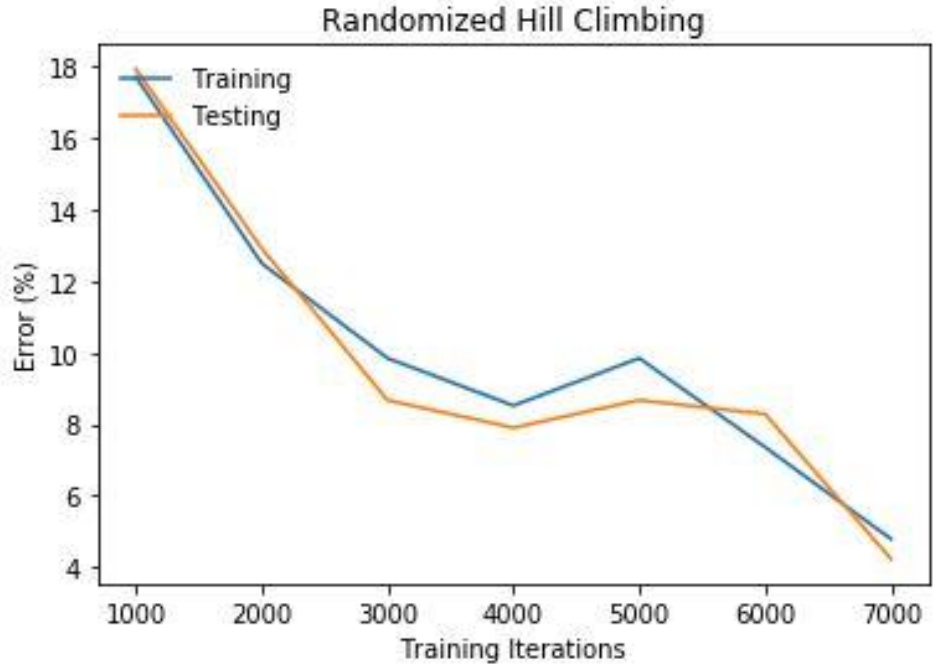

Figure 1. Randomized hill climbing was used to optimize the neural network. With each 1000 training iterations added, the percent error decreased steadily. Since adding iterations has a similar effect to increasing the number of random restarts, we can see that as the number increases, RHC was more likely to converge on the global maximum. I probably should have trained on more iterations to observe when the algorithm would converge, but its performance was already showing promise by the 7000th iteration. However, although not depicted graphically, the time it took to train the model increased exponentially with each additional iteration, which may not be desirable in some contexts.

As observed within the data for RHC in *Figure 1,* randomized hill climbing performed well on the cars dataset, with error decreasing at a steady rate, spiking slightly at the 5000th iteration before falling again. When I was analyzing this dataset using other supervised learning techniques, I observed that it was quite noisy, and many techniques were unable to achieve such low error. However, RHC is quite a robust algorithm, continually optimizing until a global optimum is reached or until the algorithm stops outperforming itself. Finding the most optimal solution is merely a question of how many iterations RHC is run, which is analogous to number of random restarts. The more iterations there are, the more likely RHC is going to happen upon the global optimum. This conclusion is represented well by *Figure 1.*

Simulated Annealing

Simulated Annealing in theory should perform better than randomized hill climbing since it is more likely to escape getting stuck in local maxima. The simulated annealing algorithm slowly decreases the temperature value, making larger jumps that may not always land in an

optimal spot but are helpful in escaping local optima early on. The speed at which the temperature value decreases in based on the cooling ratio, with a higher ratio meaning slower cooling. I ran simulated annealing twice, keeping cooling rate constant in the first run and temperature constant in the second one. The results I obtained are as follows:

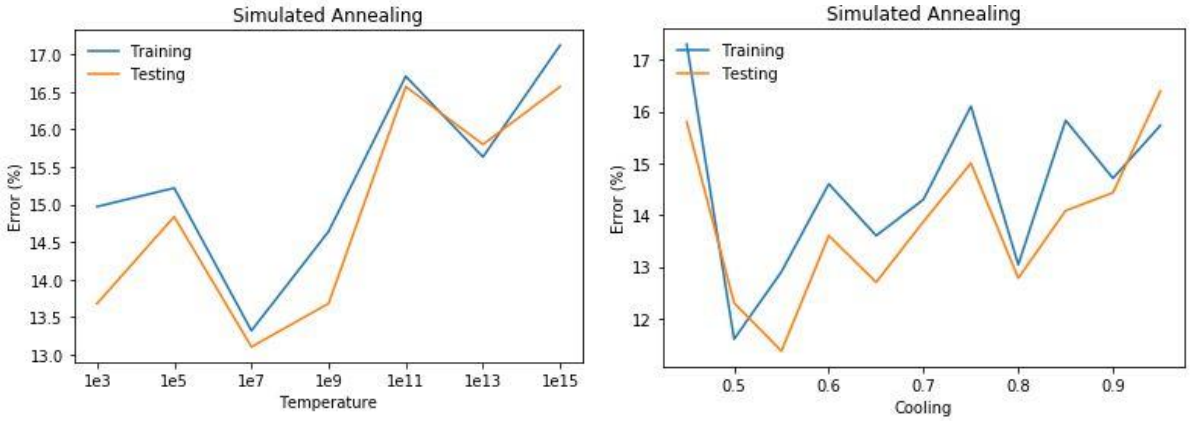

Figure 2.(a and b) The graph on the left depicts rising temperature versus percentage error, and the second graph on the right depicts rising cooling rates versus percentage error. It is worth noting that the testing error was lower than the training error in both cases, and in many of the cases that follow. This is most likely due to underfitting and the small size of the dataset. Given the small size, I probably should have changed the split in favor of training data. In both of these tests, I kept the number of training iterations fixed at 2000.

For my first run, while altering the temperature hyperparameter, I kept the cooling rate at a constant 0.95, which was the default given in the code. With increasing temperature, error increased, with the temperature 1e7 being the exception. While this may have been an accident, I used the value 1e7 as my optimal temperature in my later runs of the algorithm. The larger the temperature was, the more time it took for the algorithm to complete. Increasing temperature also leads to high probability of choosing a worse new solution with each iteration, according to the acceptance probability function. With higher starting temperatures, the algorithm was more likely to jump to another random spot early on. Given that the cars dataset is known to be noisy, too much jumping may have actually caused simulated annealing to get stuck in a local maxima.

For my second run, while altering the cooling rate, I kept my temperature at a constant 1e5, the default parameter provided in the code. Performance is more erratic, but the general trend is of increasing error percentage. The first data point incurs high error, most likely because the cooling rate was too low. Similar to my analysis for increasing temperature, error increases while cooling rate increases due to the high noise profile of the dataset and the higher likelihood of being stuck in a local maximum if too much jumping is performed early on. High temperatures and high cooling rates lead to more probabilistic jumping early on the algorithm, which in theory is an excellent way to discover global maxima and escape local maxima; however, this was not reflected in my dataset. The explanation for the high spike at a cooling rate of 0.45 is that the cooling rate was too low at that point, causing simulated annealing to act like a normal hill climbing problem and get stuck in a local maximum.

Genetic Algorithm

The genetic algorithm is an interesting algorithm based off biological evolution and the idea of natural selection. With each iteration, a new child population is created by mating two data points, mutating a data point, or simply carrying over a data point if it is good enough. While this algorithm works for many complex cases, it is very computationally intensive, taking up to two thousand seconds to run with a small number of iterations.

I ran the genetic algorithm twice, holding the mutation ratio at a constant 0.02 for the first run and holding the mating ratio at a constant 0.02 for the second run. During both runs, training iterations were fixed at 20 due to the computational complexity of the algorithm, and population size was fixed at 200. My results are as follows:

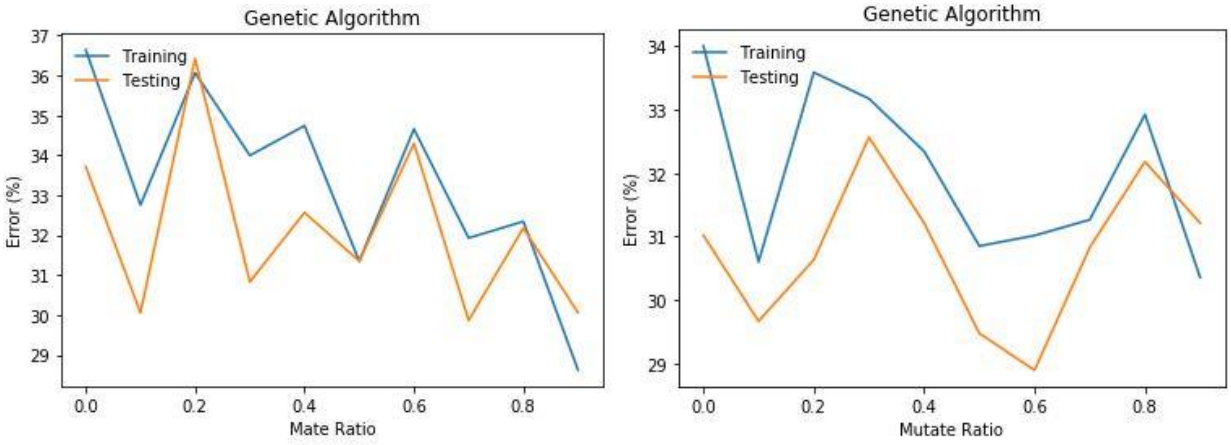

Figure 3.(a and b) The graph on the left depicts a changing mating ratio and constant mutation ratio. The graph on the right depicts a changing mutation ratio and constant mating ratio. Behavior is rather erratic, but the overall trend is that of decreasing error. The training iteration number I used was also rather small since the machine I was using to run the algorithms was unable to process more iterations given the computational complexity of the genetic algorithm. However, had I increased number of iterations, performance most likely would have improved. Thus, the errors produced from the genetic algorithm are much higher than those of randomized hill climbing and simulated annealing.

Mating in the genetic algorithm involves creating a child data point that is based off combining features of two parent data points. This can sometimes lead to data points that are very far off from the original parents, which, similar to simulated annealing, can move the algorithm away from getting stuck in local maxima and towards a global optimum. However, given the unpredictability of how the child data will turn out, performance can vary. For example, a child data point could potentially be a combination of unhelpful attributes of the parents, leading to large leaps in counterintuitive directions. This is represented by the highly volatile nature of error experienced by the cars dataset when changing mating ratios (*Figure 3.a).*

Mutation in the genetic algorithm involves randomly changing some features in a parent data point to produce a new child data point. The behavior is not quite as erratic as changing

mating ratios since the leaps are not as large. However, there were still quite a few dips and spikes in error while tuning the mutation ratio.

Given the noisiness of the cars dataset and the fixed small training iteration number, the genetic algorithm was unable to perform as consistently as simulated annealing and randomized hill climbing. The algorithm also gave way to heavy underfitting, again most likely due to the small dataset size, inappropriate test and training data split, and small iteration number.

Conclusion for Optimizations on the Cars Dataset

In a final run, I chose the optimal hyperparameters for simulated annealing and genetic mutation based off my previous data and plotted all three problems together with respect to number of training iterations.

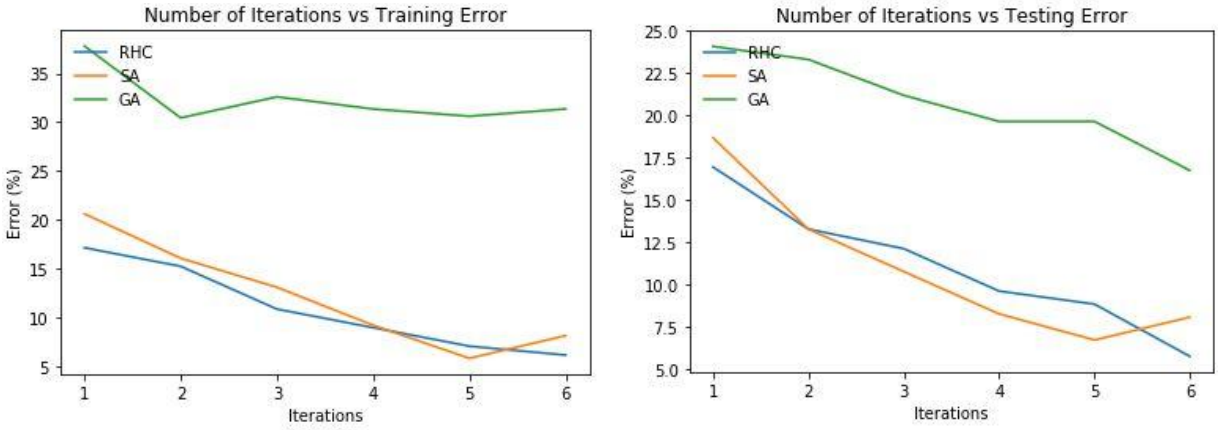

Figure 4.(a and b) These are graphs plotting training and testing error for each optimization problem with respect to training iteration number. Notice that iterations is labelled from 1 to 6. For simulated annealing and randomized hill climbing, this number is multiplied by 1000, but for the genetic algorithm, the number was only multiplied by 10. Thus, at iteration 3 labelled on the graph, RHC and SA were run on 3000 training iterations while GA was run on 30. This is due to the high computational complexity of the genetic algorithm and my machine's inability to handle it. Simulated annealing was run with temperature being 1e7 and cooling rate being 0.5. The genetic algorithm was run with mating ratio being 0.3 and mutating ratio being 0.6.

Overall, randomized hill climbing performed the best out of the three problems, though simulated annealing was close behind. Again, with increased iterations, randomized hill climbing was able to find the optimal solution due to the increased number of random restarts matching the randomness of the data and small size of my dataset. The genetic algorithm performed extremely poorly in comparison, given the noisiness of the data but also my inability to test the algorithm on a higher number of training iterations. *Figure 5* below describes the computational complexity of each algorithm, with randomized hill climbing and simulated annealing neck and neck and the genetic algorithm taking much longer to perform despite its smaller iteration number due to its nature of population redistribution.

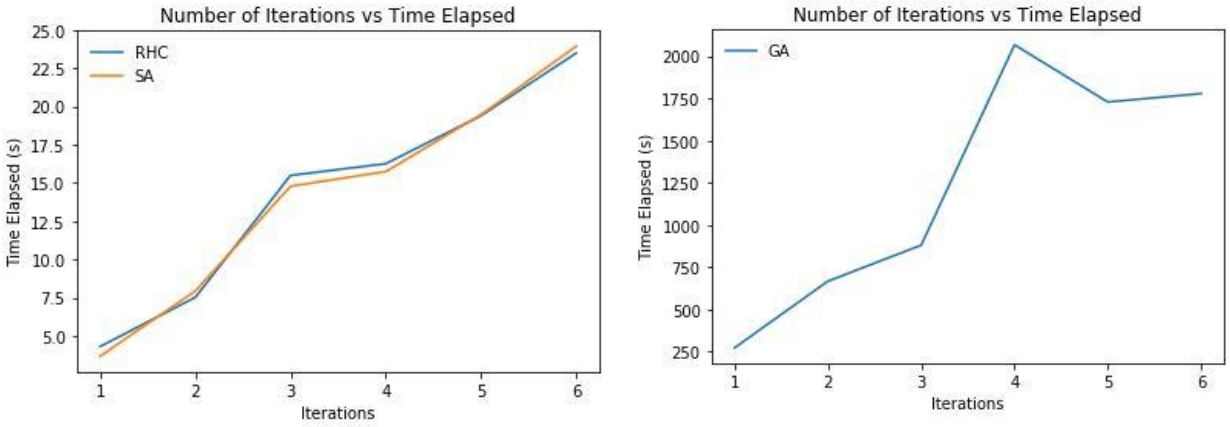

Figure 5.(a and b) Graphs depicting time elapsed versus the number of iterations. It is worth noting that the iteration numbers for *a* are scaled by 1000 while the numbers for *b* are scaled by 10. In addition, note the large difference between time elapsed between the three algorithms. When plotting both the GA and RHC and SA on the same graph, the increase in RHC and SA could not be seen due to the scale of GA.

Had I been able to run the genetic algorithm on my dataset more efficiently on a better machine, the genetic algorithm might have been able to converge on a better solution than randomized hill climbing and simulated annealing, since the nature of the genetic algorithm is for it to perform better on more complex problems. However, I was not able to test this hypothesis given my physical constraints.

Again, given the noisiness of the cars dataset, optimization problems such as randomized hill climbing and simulated annealing performed better than simply running a neural network with optimized hyperparameters via backpropagation like what I did in the supervised learning project.

Optimization Problems

The optimization problems used to describe simulated annealing and the genetic algorithm chosen were the 4-peaks problem and the count ones problem. For each problem, I still ran randomized hill climbing as a nice basis for comparison to simulated annealing since it does not have any noteworthy tunable hyperparameters.

The 4-peaks Problem

The 4-peaks problem involves bit strings and counting the numbers of consecutive ones and zeros starting from the front and back respectively. The problem is parameterized by a value T that gives the fitness a reward of 100 if the number of ones or zeros is over the threshold T. Given this, the 4-peaks problem has two global optima, one where the number of ones is greater than T or the number of zeros is greater than T. The problem also has two local optima, where all the bits in the bit string are either one or zero. In theory, a normal hill climbing algorithm could very easily get stuck in these local optima if placed in an unlucky spot.

Figure 6 depicts the fitness versus the number of iterations performed for each algorithm. I tuned some of the hyperparameters to get my results, which I detail inside of the figure description. It's also worth noting that the 4-peaks problem is designed to work optimally with the genetic algorithm, which I will explain later.

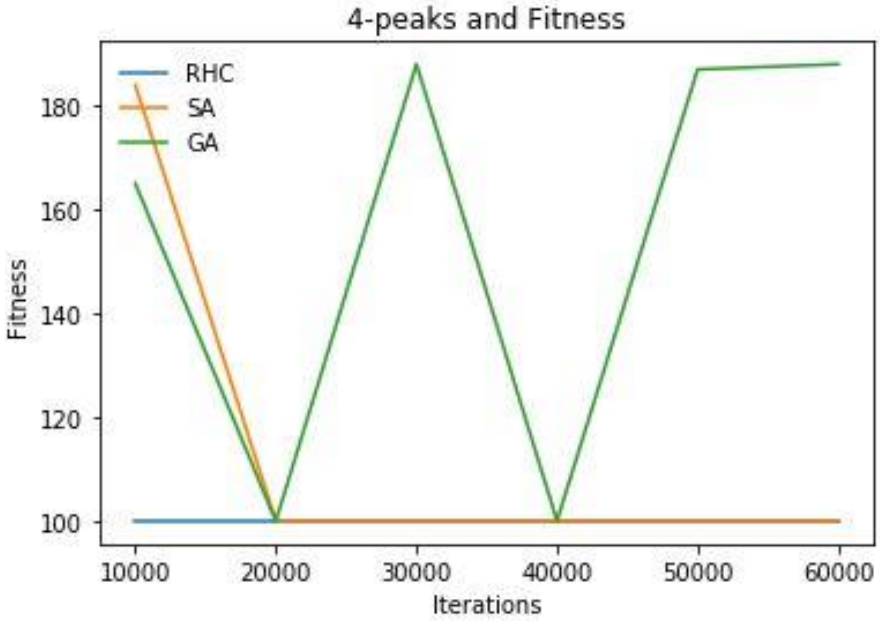

Figure 6. The graph developed from running randomized hill climbing, simulated annealing, and the genetic algorithm on the 4-peaks problem. The bit strings were set to 100 bits and T was set to 11. The optimal temperature and cooling rate for simulated annealing were 1e11 and 0.95 respectively. The optimal mating and mutation ratios for the genetic algorithm were 160 and 20 out of a population of 200.

As observed from *Figure 6,* randomized hill climbing immediately get stuck at a local optimum, and simulated annealing initially starts close to a global optimum before taking an unfortunate step in the wrong direction. Hill climbing algorithms like RHC and SA are more likely to get stuck in a local optimum because of their nature to optimize on the parameter that already has a higher value. In this example, I set the bit string to be 100 bits and the threshold to be 11 bits. This means that the global optima are the cases where 11 bits are zeros and 89 bits are ones, or 11 bits are ones and 89 bits are zeros. Once achieving this case, a reward of 100 is added to the fitness, indicating a global optimum has been found. Due to the nature of hill climbing, if the algorithms were presented with data where there were more zeros than ones and vice versa, the algorithm would increase the value of the bit with more data points. For example, if hill climbing were presented with a string that had 4 zeros and 40 ones, it would optimize the number of ones until there were 100 ones, making the problem hit a local optimum and get stuck. In *Figure 6*, this is demonstrated by how RHC and SA both flatline at 100, the local optimum.

On the other hand, the genetic algorithm managed to find its way towards the global optimum, which in this case is 189, given that an optimal solution would be 11 ones and 89 zeros or vice versa. 100 was added as a reward to the fitness, hence the genetic algorithm's value of

189 at the end of 60000 iterations. The 4-peaks problem works well with genetic algorithms when the mating ratio is high. While mating, the algorithm is able to piece together useful parts of two parents into a child that is closer to a global optimum without getting stuck in the same spot like hill climbing algorithms do. Thus, out of a population of 200, I mated 160 and mutated 20, achieving the global optimum by the 60000th iteration. The algorithm approached the global optimum at the 30000th trial as well but dipped back down to the local optimum for a short while. This is most likely due to some chance unsuccessful mating that chose poor combinations of building blocks from two parents, but GA was able to escape out of it.

The Count Ones Problem

The Count Ones problem is a straightforward problem that behaves exactly how its name implies. The optimization algorithm simply needs to count the number of ones in a bit string of N length. There are not many local optima and only one global optimum, where the optimization algorithm can produce an input that has the maximum number of ones. The following graphs were the output of running each optimization algorithm on the count ones problem:

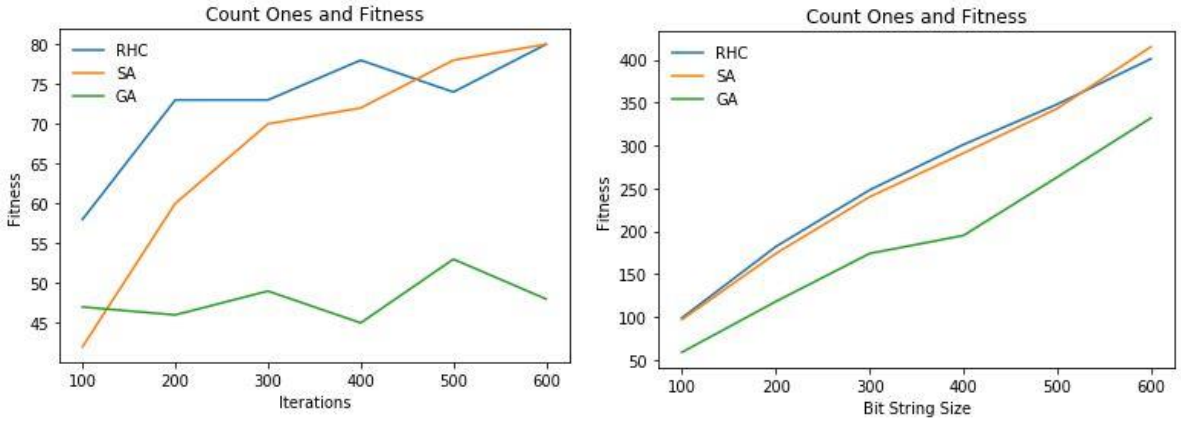

Figure 7.(a and b) I ran the three optimization algorithms on the count ones problem, with graph *a* depicting algorithm performance in relation to the number of iterations performed and graph *b* depicting algorithm performance in relation to the size of the input bit string N. For graph *a*, N was fixed at 80. For both graphs, after tuning parameters, I fixed the temperature and cooling rate for simulated annealing and the mating and mutating ratios for the genetic algorithm. The temperature and cooling rates were set to 1e4 and 0.9 respectively, and the mating and mutating ratios were set to 3 and 16 respectively for a population of size 20. For the second graph, iteration number was fixed at 600, seeing as how based on *a*, we can see that some algorithms were able to reach the global optimum by the 600th iteration.

We can immediately see that simulated annealing (which is similar to randomized hill climbing) outperformed the genetic algorithm for all iterations and sizes of N (input size). This is because there is not really a local minimum that hill climbing algorithms can get stuck in, so simulated annealing's tendency to hop towards the global optimum and converge early on pays off. However, the genetic algorithm has a harder time with this straightforward problem, partially

due to the fact that genetic algorithms are designed to work better on more complex problems. Mating and mutating different inputs did not allow the genetic algorithm to approach the simply defined global optimum.

Conclusion for Optimization Problems

The 4-peaks problem and count ones problem were able to distinguish some key differences in the strengths and weaknesses of simulated annealing and genetic algorithms. The 4-peaks problem had very prominent local optima, making hill climbing algorithms like simulated annealing very susceptible to getting stuck. Since one of the only ways to make it out of a local optimum was to take counterintuitive steps (e.g. in a scenario where there are 4 ones and 40 zeros, reaching a global optimum would actually involve adding weight to the ones even though the value is technically less), simulated annealing is not likely to escape from the local optima. The 4-peaks problem is a more complex problem to solve than count ones, meaning that while hill climbing algorithms suffer, the genetic algorithm thrives.

Thus, when the algorithms are presented with a simple problem like count ones, simulated annealing performs superbly while the genetic algorithm suffers. To begin with, cross over functionality is not very helpful when the problem is as simple as counting ones in a string. The genetic algorithm could keep mating data points to get new children that might not be anywhere close to the global optimum or mutating in random ways that do not reach the global optimum, and thus be unable to find its way to that optimum. On the other hand, with the simplicity of the problem, simulated annealing can quickly find its way towards the global optimum and climb with incredible speed until it reaches the top.

In conclusion, the most prominent discoveries from these two optimization problems is that hill climbing algorithms like simulated annealing and randomized hill climbing work the best on simple optimization problems whereas the genetic algorithm works the best on optimization problems with more complex solutions. For simulated annealing to work better on more complex problems, perhaps the algorithm could be changed so that its nature to make random jumps is a bit more prominent even towards the end of the function’s lifecycle, and for the genetic algorithm to work better on simple optimization problems, perhaps the split of parent attributes in a child could be attributed to some weight that could be placed on the parents to emphasize which attributes are closer to an optimal value.