Peter Lee
11/6/2018
Cs 4641
Project 2

Summary:

For the first part of my project I used a Machine learning library called ABIGAIL to implement a neural network where instead of traditional backpropagation, the weights were altered with Random Hill Climbing, Simulated Annealing, and a Genetic Algorithm. Honestly I was a little confused as to why we were doing this as the current literature has drifted away from exploring such options however I now realize the importance in gaining a foundational understanding of these algorithms. I am using my Breast cancer dataset that I used in Assignment 1 since it's a binary classification problem with 30 post-processed attributes taken from an image. The data is stored in 'breastData.csv.'

For the second part of my project, I chose 3 optimization problems: Flip Flop, Count Ones, and Traveling Salesman. Each of these problems are very different from one another in terms of how their local-minima are set up as well as NP hardness. Overall I expected Genetic algorithm to perform the best at NP-hard problems and Simulated Annealing and Random Hill Climbing to do well in simpler problems.

Random Hill Climbing iteratively climbs to the highest point based off of random start points. Hill climbing tends to be very greedy and gets stuck in local optima however it is computational easy and very little memory and processing are needed to implement it. I used ABIGAIL to use RHC to optimize a neural net.

Simulated Annealing implements Random Hill Climbing however it also includes a probability that it will 'jump' to a worse solution, this is to 'shake' the algorithm away from local minima that a greedy RHC can get stuck in so that a simulated annealing algorithm can try new locations to optimize itself. It's a bit more computationally intensive than RHC but it's still much leaner than Genetic Algorithm performance-wise.

Genetic Algorithm creates a population that mate and mutate to create a constantly changing population. The fittest of the population survive to mate and mutate some more. Fitness is determined by an objective function and the rates of mutation and mating are set by the user. Genetic Algorithm typically are very computationally intensive however they outperform RHC and SA in NP-hard problems (such as human life).

Neural Network Optimization

Randomized Hill Climbing For Neural Network

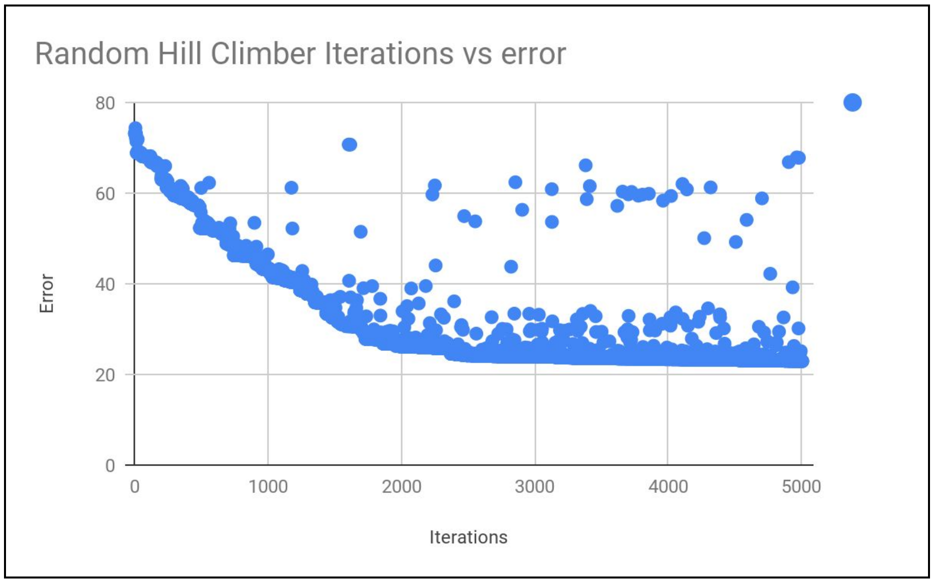


Fig 1, Random Hill Climber over Iterations

Randomized hill climber was implemented on the test data. Relative to the other algorithms, It performed pretty poorly to my dataset. This was most likely due to it being stuck in local optima. You can see the overall trend decreases as well as random instances where the hill climber tries another point that are suboptimal.

Simulated Annealing For Neural Network

Simulated Annealing within the ABIGAIL problem had two parameters for tuning. Initial Temperature as well as cooling. Temperature relates to how likely the algorithm is to pick a suboptimal solution for the next step whereas cooling is the change of temperature over iterations. Ideally you'd want to pick a relatively high temperature initially to get out of local minima and cool over time so that you converge to a near-optimum solution. I tried altering both of these parameters with some encouraging results over my breast cancer dataset using ABAGAIL.

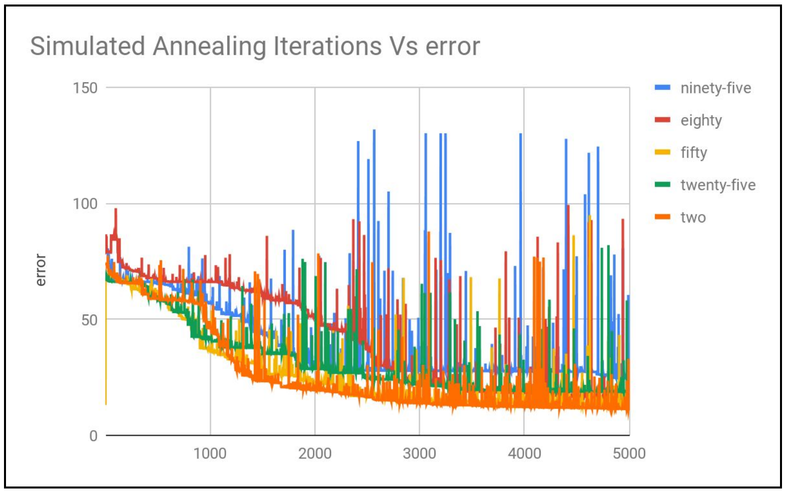


Fig 2, Temperature held constant at 100, changed cooling rate

Fig 2 showed that a high cooling rate overall hurts your performance over time and a lower cooling rate is better at converging more quickly. This makes sense since a lower cooling rate means that the algorithm will stay “hot” longer meaning that it's more likely to make riskier decisions that may lead to better solutions.

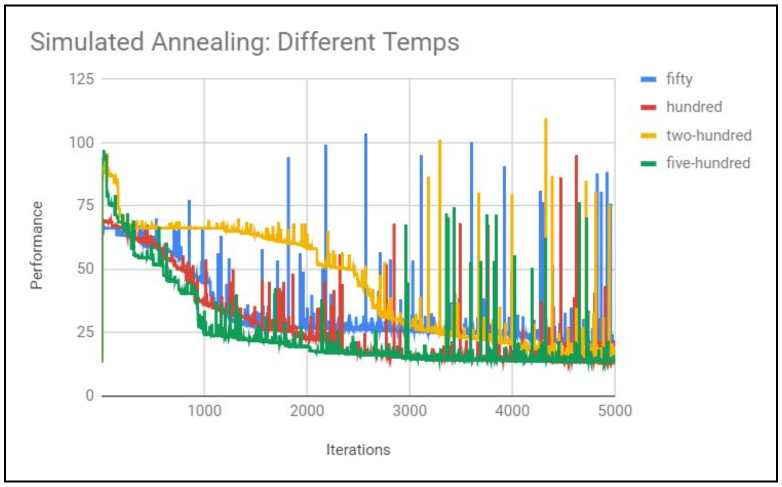


Fig 3, changed initial temperature, the mutation rate was held constant at .5

Fig 3 showed that a higher initial temperature leads to a steeper learning rate. This makes sense since a higher temperature means that the algorithm is much more willing to try out different solutions even if may hurt performance, leading to the possibility for improvement. Overall the results both numerically and graphically show that you want a low cooling rate and a high initial temperature. Of course with anything, modesty is key. For simulating annealing you want to ensure that you’re algorithm cools appropriately or else the algorithm may take a long time to converge and cool and this may cost a lot of computational time. You can increase the performance of your Simulated Annealing by cooling slowly while also having a high Initial heat, Ideally this would lead to a much faster convergence and therefore less iterations have to be run.

Genetic Algorithm for Neural Network

Below I tuned a genetic algorithm, the first parameter is Mating population, the second parameter is the mutating population. The overall Population was 100.The general trend appears that as you increase the amount of mutation and increase the mate rate, the minimum error decreases. This is most likely due that the function is able to tease out the best attributes to look at and tune. However if the mutation rate and mating rate is too high, the problem may be the reverse as we can see in the performance of (50,50) in **Table 1**. Because the mating is too high, genetic diversity may be driven down as only the most fit for that iteration survive however of the population that was eliminated, some useful information may have existed which could improve the overall performance. In addition if the mutation is too high convergence may not occur as the population may find the true solution however it would be quickly become sub-optimal with the introduction of mutations if the mutation rate is too high. Like always, Moderation is key when it comes to tuning.

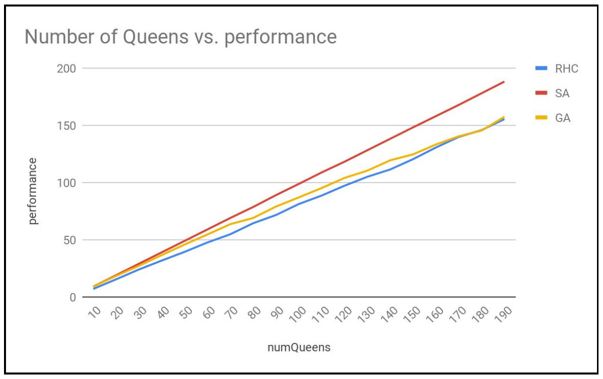
(Mate, Mutation)	(10,10)	(10,20)	(10,30)	(20,10)	(20,20)	(20,30)	(30,10)	(30,20)	(30,30)	(50,50)
num iter	55265	75080	106501	92951	113689	136170	128886	140801	170867	128774
Min error	15.637	16	16	16	13.92	16	16	14.7622	12.6011	18.5

Table 1, Mate and Mutation Tuning table for Genetic Algorithm.

Optimization Problems

Flip Flop

Flip flop is a problem where each bit in a bitstring alternates between 1's and 0's. For example a bitstring of '0001' would have a value of 2 since two different bits were near each other while '0101' would have a value of 4. Although the solution seems intuitive for a human to solve, machines have a bit of trouble with this problem and there are many local optima that the machines may get stuck in.



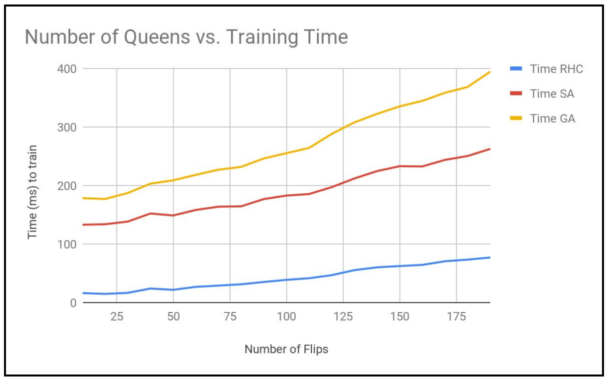
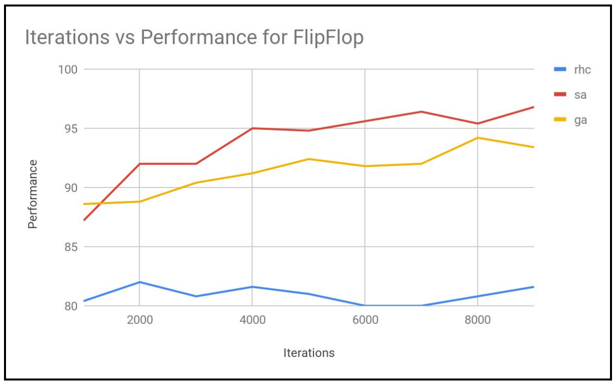


Fig 4. run with 200,000 iterations. A total of 5 times.

The Best performance was from Simulated Annealing. This is most likely due to the fact that Simulated works very well in its 'neighborhood' and is willing to take risks when its temperature or its likeliness to make 'bad' decisions is higher so It's able to better hop out of local minima than Random Hill Climber (RHC). Genetic algorithms and RHC suffered as they got stuck in local minima that they were unable to escape from.



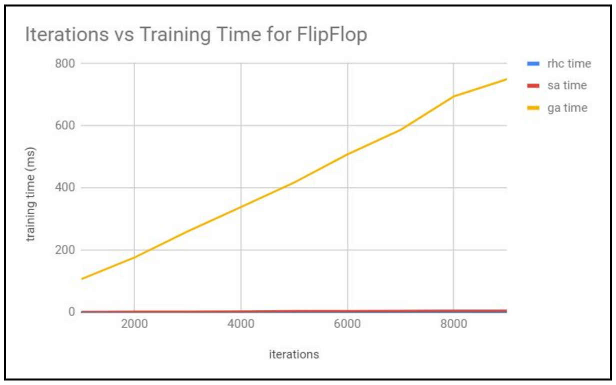


Fig 5. Iterations and Performance with 100 Queen, average of 5 times.

RHC does however have the advantage that it found solutions incredibly quickly compared to the rest, Simulated Annealing did slightly worse than RHC and Genetic Algorithms did the worst in terms of time-performance by a lot.

Count Ones

The Count Ones problem was chosen because it's both intuitive for a human and easy for a computer to solve since the global maximum is easy to approach. The problem also involves bit strings except this time the maximum number of 1's in a bitstring is the optimal solution, therefore altering a bit to '1' from a '0' will always be the correct decision such that local minima aren't really a problem since the problem is straightforward i.e. find every '0' and flip it. Unsurprisingly the simplest and most greedy of the algorithms, Random Hill Climbing won out. This was most likely due to the fact that RHC converged quickly to the global maximum whereas Simulated Annealing took hits for trying suboptimal solutions which wouldn't help in this case. Genetic Algorithm also suffered most likely since mutation's where swapping position of bits occurred, led to no increase in score.

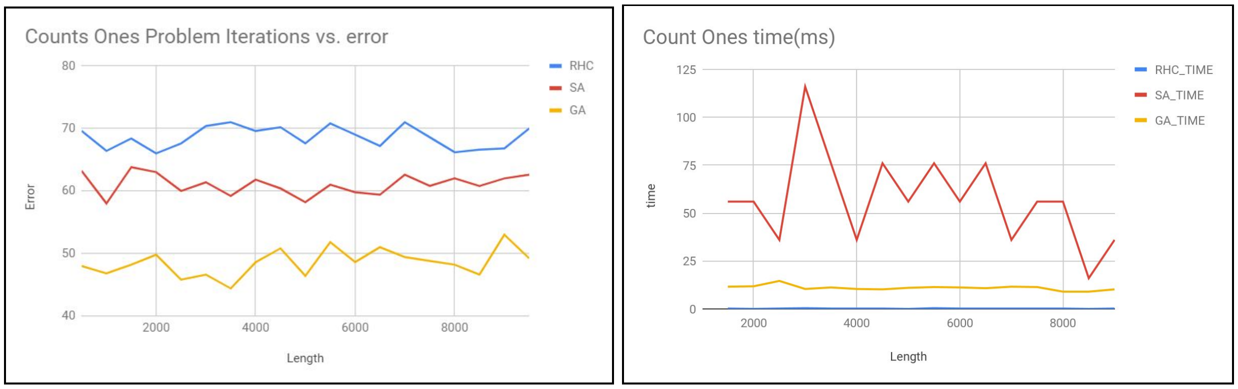
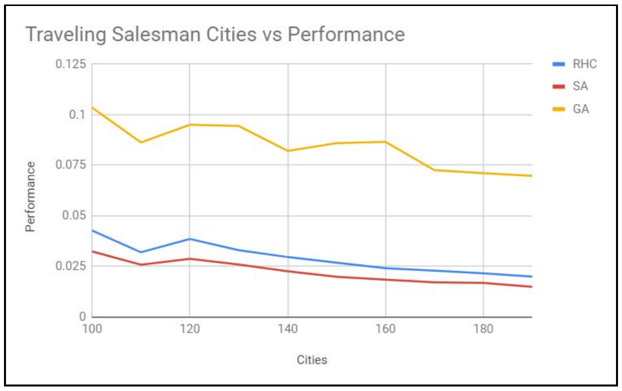


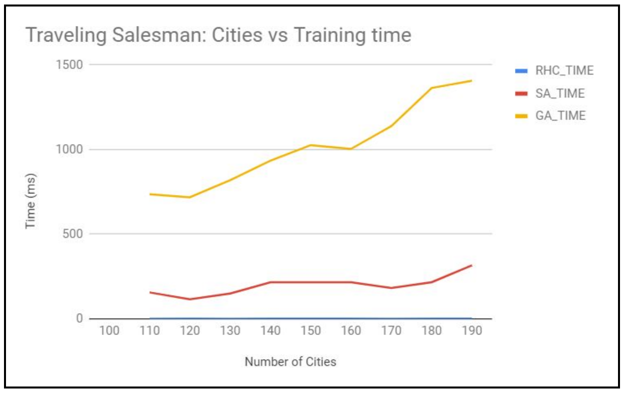
Fig 6. Count Ones performance over Length and training Time.

Overall if you look at Fig 6, you can see RHC did the best, Simulated Annealing took 2nd place, and Genetic Algorithm did the worst. When we look at training time, Simulated Annealing did the worst, Genetic Algorithm took 2nd place and RHC was the fastest.

Traveling Salesman Problem:

I implemented this because I knew this was an NP-Hard problem and I predicted that the Genetic Algorithm would outperform the RHC and SA methods.





The Traveling Salesman problem is an np-hard problem and its complexity increases dramatically. Genetic Algorithms perform exceptionally well as the principle is that good general solutions tend to propagate while bad solutions tend to die. In the case for genetic algorithms, parents with short paths will propagate whereas more complex maps will die out. In addition parents with short paths are likely to mate and produce children with even shorter paths, this kind of learning is essential to solve such np-hard problems. Simulated Annealing and Random Hill Climbing suffered as they were stuck in local minima, even though a solution in the local state may be ideal. In terms of time spent searching, Genetic Algorithm took the most time, Simulated Annealing was in the middle, and RHC was fastest.

Optimization Conclusion:

Here is an overall summary of what I've found through these optimization problems. These algorithms once again demonstrates the idea of "No Free Lunch" where different kinds of optimization problems outperform one another in different Domains.

	RHC	SA	GA
Pros	Simple, easy to implement fast training. Great for problems with little underlying structure.	Easy to implement, fast training slightly better than RHC at most problems (especially when theres a few local minima).	Great for NP hard problems.(such as traveling salesman)
Cons	Greedy (gets stuck in local minima). Bad at NP Hard Problems	Can get stuck in local minima. Bad at NP Hard Problems	Takes a long time to train.

Citations:

https://www.researchgate.net/post/Why_is_the_mutation_rate_in_genetic_algorithms_very_small

ABAGAIL Library : https://github.com/pushkar/ABAGAIL

Breast Cancer Dataset: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

Traveling Salesman:
https://optimization.mccormick.northwestern.edu/index.php/Traveling_salesman_problems