Random Optimization

Tiera Lee

ABSTRACT

This paper is an exploration of the random optimization algorithms randomized hill climbing, simulated annealing, and genetic algorithms. The first section compares the performance of these on algorithms on the Bach chords dataset from a previous paper. The second section seeks to highlight the strengths and weakness of each algorithm by comparing their performance with different optimization problems. The third section explores the effects of mutation rates of crossover techniques on genetic algorithms via building neural networks playing the Flappy Bird game.

Introduction

Random optimization is a family of numerical optimization techniques that attempts to solve for the optimum solution of functions and does not require a gradient. When applied to machine learning RO methods search for the best hypothesis that will maximize the fitness function, which determines how "good" a solution is.

I utilized the ABAGAIL library for the implementations of randomized hill climbing (RHC), simulated annealing (SA), and genetic algorithms (GA) in sections one and two

Section I. Implementing Random Optimization Algorithms of Neural Nets

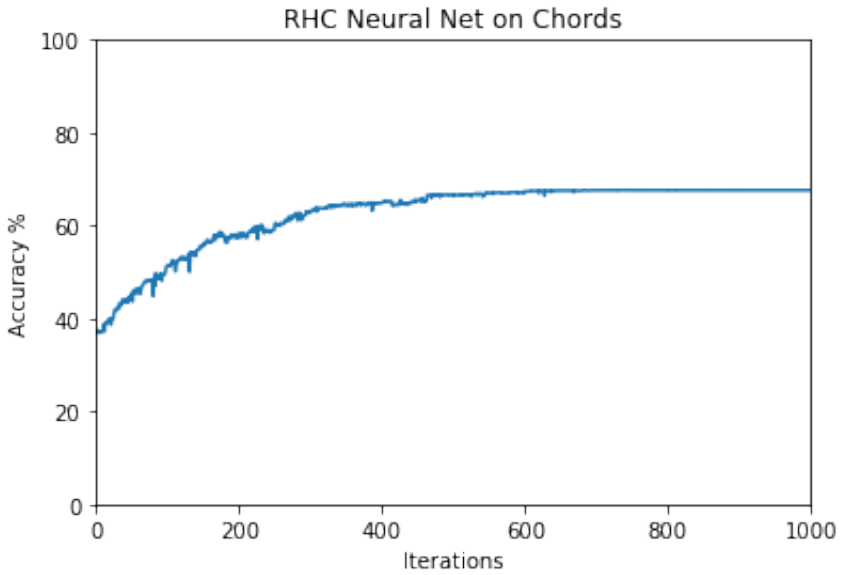
For this section I applied various random optimization algorithms find optimal weights for a neural net on the chords data from project 1. This neural net will attempt to classify the type of chord that is being presented in a Bach choral. The chords classifications available are major, minor, and diminished. The baseline for accuracy is 67% due to 67% of the data being in the major class. Most implementations of neural nets in project 1 has testing accuracy of 67%, with only the neural lets with high momentum performed better. Just like project 1, the neural network constructed has 14 input nodes, one hidden layer of 9 nodes, and 1 output node.

I applied RHC, SA, GA with various parameters access performance. The RHC and SA curves presented in the following plots are the average of 5 runs and GA curves are the average of 3 runs. Only the learning curves for accuracy are presented in this

section. All of the above algorithms are evaluated with ABAGAIL's respective implementation.

1.1 Randomized Hill Climbing

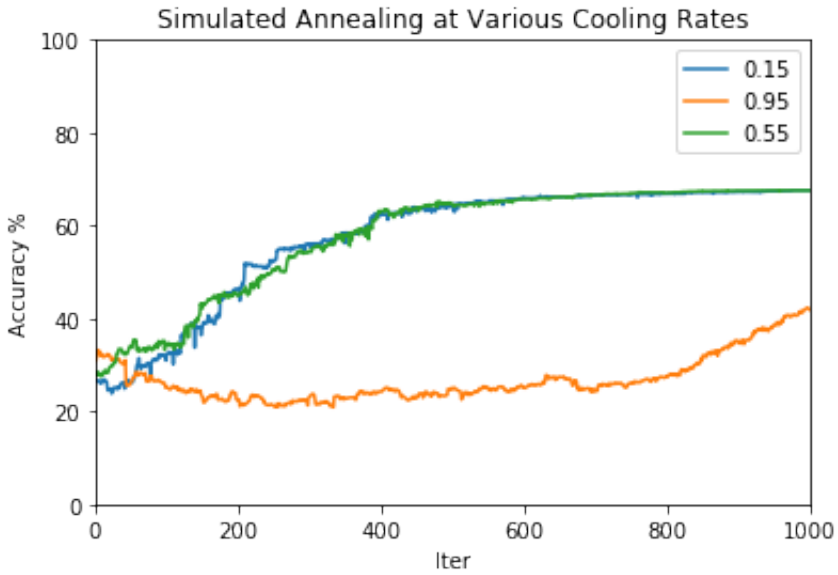
RHC performs like a greedy algorithm, always looking for the next best answer given its current environment. This makes it adverse to immediate "bad" decisions, even though it can lead to global optima in the future.



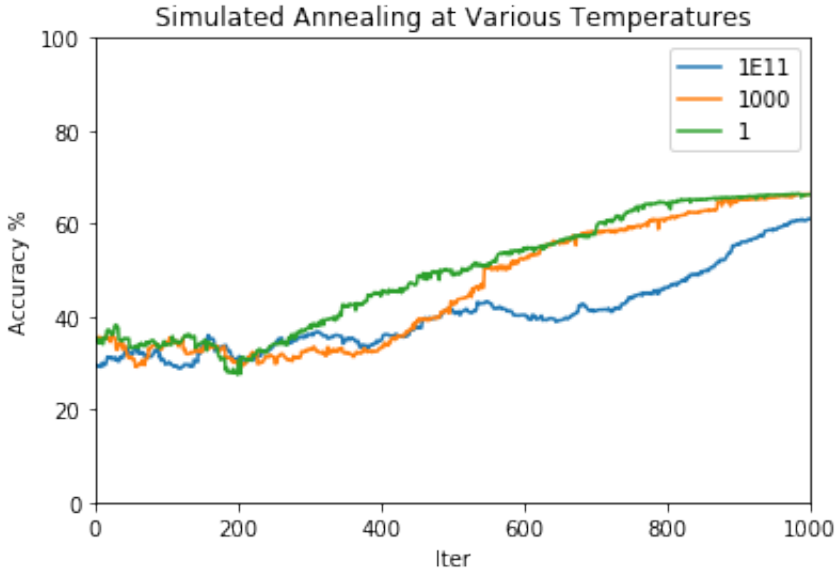
The training accuracy converges to 67%, the baseline of a random guess. This makes sense because there is a large basin of attraction under solutions that classify the data as being part of the majority class. If a larger optima existed, the chances are of RHC converging to it is much lower.

1.2 Simulated Annealing

SA works very similar to RHC except it has more opportunity to exploit and explore the space due to its temperature and cooling properties. It is more likely to explore immediately bad decisions. I explored how altering the starting temperature and cooling rates affect convergence.



The cooling rate of 0.95 performed significantly worse than lower cooling rates, but seems to begin to move towards convergence after 800 iterations. The faster the temperature cools, the less space that can be exploited and explored. If SA is unlucky enough to start in a suboptimal solution, then it is more likely to get stuck as it cools faster. Just like annealing in physical swords, if you cool down the sword too quickly, it cannot find the best atomic configuration, and it won't be as a strong. The cooling rates of 0.15 and 0.55 have close performances and converges about after 400 iterations. The accuracy converges to 67%. Like RHC, SA does not perform optimally when there are large basins of attractions. Choosing everything to be the majority class has the largest basin of attraction, even if it might not be the global maxima.

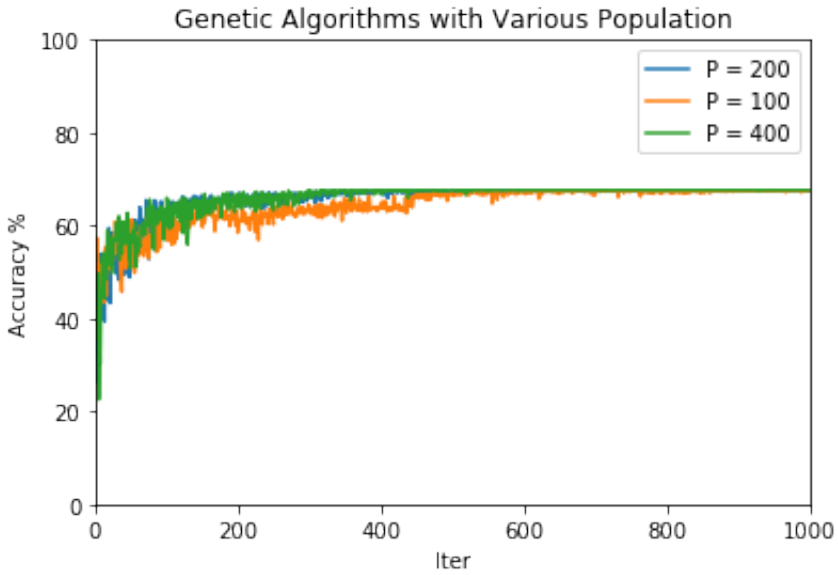


The various starting temperatures start to diverge after about 200 iterations, with a starting temperature of 1 converges to the optima faster, while 1e11 takes a longer time. This initially surprised me, but makes sense given with the given dataset. A

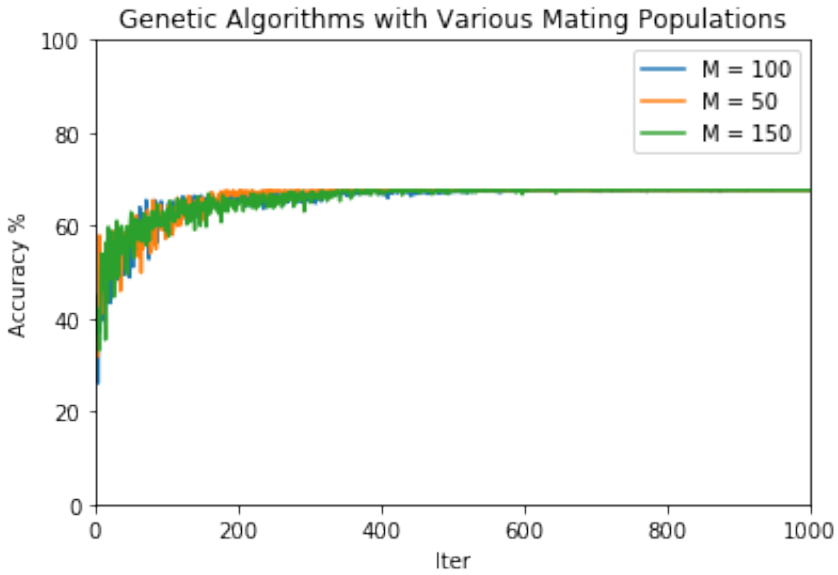
higher temperature will take more iteration to converge to the optimal solution in order to explore the entire space. Lower starting temperatures will explore less, and converge to whatever local maxima it's closest to. This dataset is highly imbalanced, and as stated before, the basin of attraction for choosing one class is large. Lower temperatures will just find these optima faster since it's likely to start at this point anyway.

1.3 Genetic Algorithms

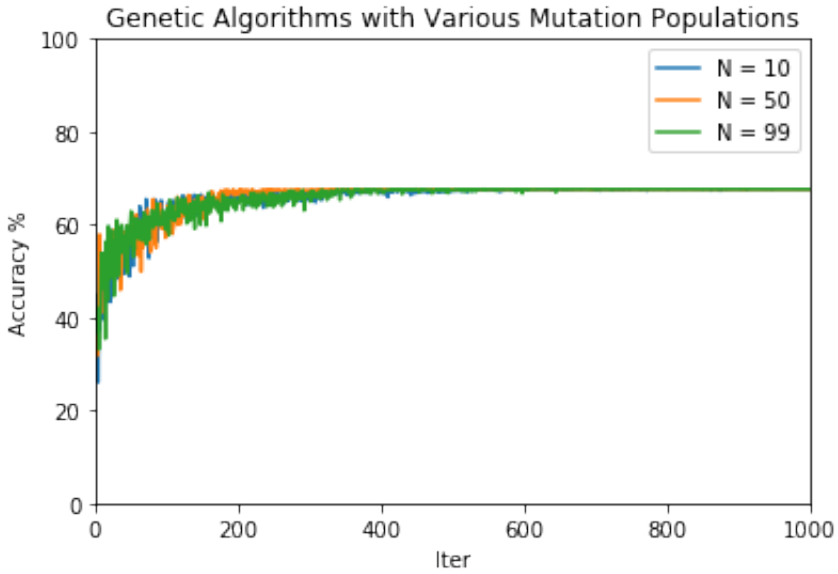
Genetic algorithms are modeled after genetic evolution in biology. First start off with a population of hypotheses represented as bit string. Select the hypotheses that have the highest fitness function performance, have two of these fit hypotheses "mate" by crossing over (swapping) sections of their bit strings, and probabilistically mutate points in these new hypotheses. Repeat this method until the optimal solution is achieved. I experimented with changing the population size, mutation and mating populations.



The lowest starting population of 100 took longer to converge to the optima, while populations of 200 and 400 performed similarly. Smaller populations will take longer to converge simply because the initial population set is less likely to contain the optimal solution, and there are less cross over combinations.



The population set is 200 hypotheses, so a mating population of 50, 100, 150 corresponds to a 25%, 50%, and 75% mating percentage respectively. This did not have much affect on the solution convergence, as all the scenarios converged to 67% around 350 iterations. The dominant genetic feature of just choosing the dominant class quickly represents most of the gene pool, so most generations will contain this, and thus limit the diversity of hypotheses as time continues.



Mutation populations are in relation to a mating population of 100. Mutation of 10 is 10% of the mating population, mutation of 50 is 50% of the population, and a mutation of 99 is 99% of the population. Altering the mutation rate does not have a large affect on this data set. Like above, the dominant genetic feature is choosing the dominant class, limiting the genetic diversity of each generation, and mutations are essentially phased out.

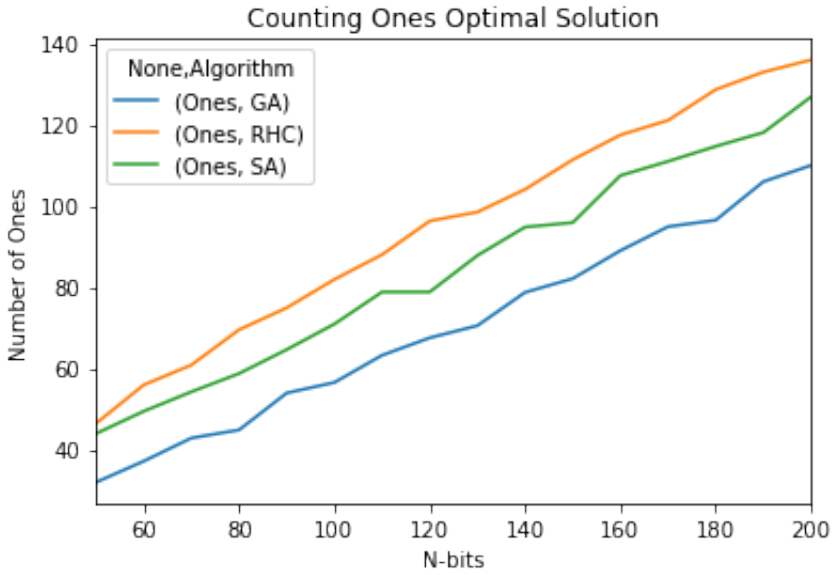
Section 2. Highlighting Random Optimization Strengths

The no free lunch theorem of machine learning can be summarized as "there is no universally good method for every problem." The person utilizing these algorithms must understand the underlying domain knowledge and choose appropriate representations of the problem they want to solve. This same theorem must be remembered when applying random optimization algorithms. Below are three optimization problems that will highlight the strengths of either randomized hill climbing (RHC), simulated annealing (SA), or genetic algorithms (GA). The underlying solution spaces for each problem are different and exploit the various methods of these algorithms.

All of the following are created using ABAGAIL's implementation of the respective optimization problem. RHA and SA run for 200,000 iterations, while GA runs for 1000 iterations. The solution curves in each graph are the averages of 10 complete runs.

2.1 Counting Ones

Counting Ones is a function that returns the number of 1's that exists in an bit string. This is easy and obvious for a human to evaluate, but takes more work for an algorithm to figure out.

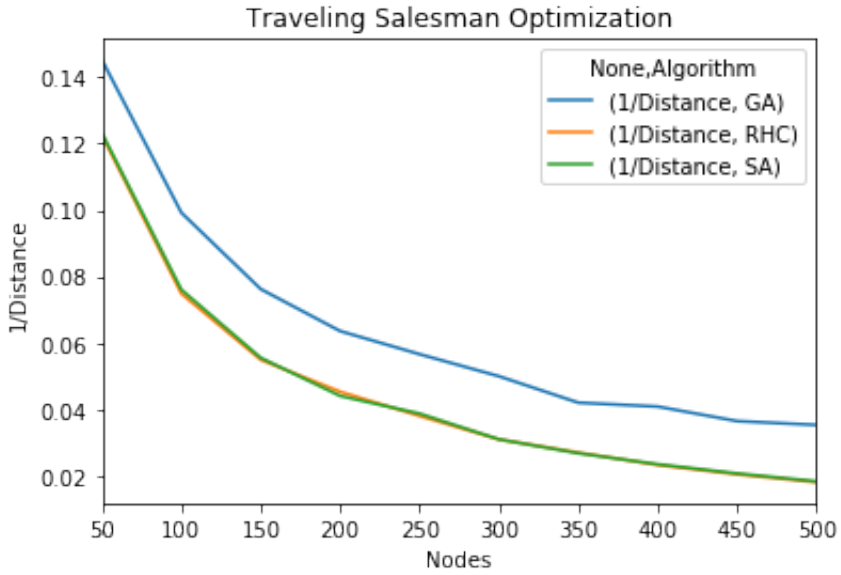


RHC is clearly the better performer. This can be explained the by space have only one maxima, which all other solutions will "climb" towards, much like an actual hill. SA explores the space too much and can get stuck in a region of the given space that is suboptimal once its temperature is low enough. For GA, mutations are not very helpful for finding the optimal solution, and can actually lead it in the wrong direction.

2.2 Traveling Salesman Problem

The traveling salesman problem (TSP) asks the following question: Given a list of cities and the distance between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

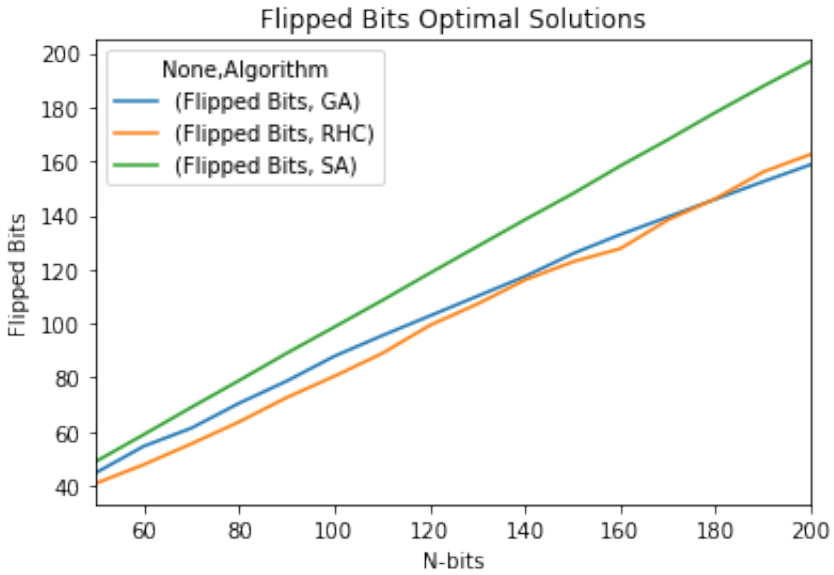
Using the ABAGAIL implementation of TSP, a random, connected graph of N nodes is generated. The fitness function for a solution is defined as 1/distance.



It is clear to see that a genetic algorithm outperforms randomized hill climbing and simulated annealing, no matter how many nodes are in the graph. This is due to genetic algorithms starting with a set of possible solutions, and continues to improve upon the solutions with the greatest fitness. RHC and SA, on the other hand, explore only one solution at a time, and essentially "forget" previously good solutions. They must take more iterations to converge to the same solution that GA does. This forgetfulness explains why RHC and SA have almost identical curves in the above plot. The inverse shape of the curves is simply do to the fact that the more nodes that exist in the graph, the greater the distance the salesman must travel to visit them all.

2.3Flip Flop

Flip Flop seeks to count the number of alternating 1's and 0's in a bit string, including the first bit. For example a bit string of "110" will return 2. The optimal solution would a bit string with constantly alternating bits, which would return a solution the same length of the string. For example "101010101010" would return 12. Since there can be relatively large sections, in relation to its length, of a bit string that does not alternate, multiple local maximums can exist in the solution space. ABAGAIL randomly generates a bit string of size N.



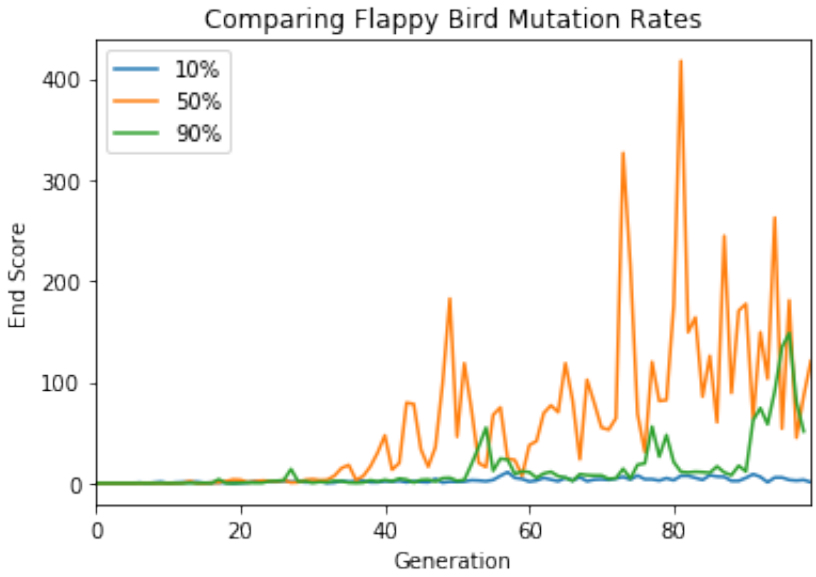
Simulated annealing consistently outperforms RHC and GA. SA performs well in solution spaces with close local optima with small basis of attractions. For bit flips, all the basins are fairly small and close other, allowing SA to escape plateaus. RHC's weakness is that it does not want to make bad decision, based off current location, and will not move past these plateaus, no matter how small. GA requires much more time to converge to the maximal solution.

Section 3. The Flappy Bird Problem

The Flappy Bird Problem is simple: How do you train a genetic algorithm to learn the weights of a neural net necessary to get the highest score in Flappy Bird. I explored the probability of mutation and various crossover techniques affected the performance of the game over on a population of 30 nets for 100 generations. Each scenario was ran three times, and their average performance is represented by the graphs below.

3.1 Probability of Mutation

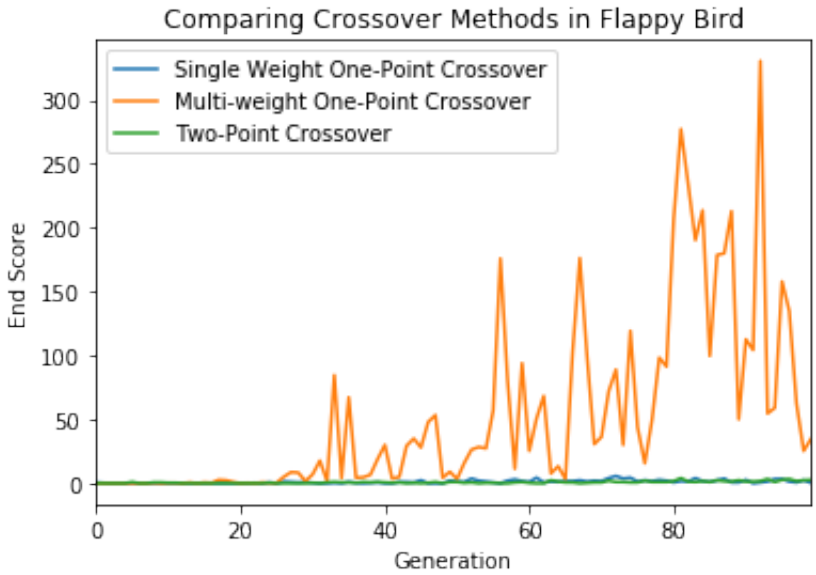
Mutation occurs when a neural net weight is probabilistically chosen, and is slightly altered. This implementation of Flappy Bird alters the weight by a value that lies within the uniform distribution of -0.5 to 0.5. The probabilities of mutation selected were 10%, 50%, and 90%.



During the 20 generations, performance is consistent among the different mutations rates because the GA is still learning. After this, the 50% mutation rate starts to out perform the 90% and 10% mutation rates. Even with the variable performance, 50% mutation rate overall performs the best. A mutation rate of 90% doesn't being to improve performance until about the 50th generation. This is because the weights are mutating so fast that the genes with the best weights cannot be passed down. The mutation rate of 10% slightly improves around the 58th generation, and it's not much better than when it started. This happens because it the original population weights are not very fit to begin with, the future generations will only increase in fitness slightly, When a good mutation, one that significantly affects performance, presents itself it can then be added to the gene pool and passed to future generations. The performance of the various mutation rates suggests there is a goldilocks range for mutations that result in optimal performance.

3.1 Crossover

Crossover is the mechanism in which two neural nets "mate", Two neural net will swap weights in the same location, to create two children neural nets. I compared one point cross over of the first half of the array of weights, two point cross over the first and last thirds of the array of weights, and a single point cross over of just the middle weight.



Two-point and single weight one-point crossovers both had minimal improvement of performance in 100 generations. Single-weight makes slight gains after 40 generations. Multi-weight one point cross over outperforms the other two methods starting around the 30th generation. The best explanation for this performance is that features associated with the first half of the weights in the array are more important, holding more relevant information. Crossing these weights takes full advantage of the GA process, thus evolving to better performing neural nets.

Conclusion

For the Bach chords dataset, all of the random optimization algorithms converged to a training accuracy of 67%, which implies the optimal hypothesis is to classify all of the instances as a member of the majority class. While GA are more sophisticated and can return optimal solutions in more complex spaces, it is not the appropriate RO for every problem. SA and RHC can perform better in certain cases. Population size, mutation rates, and crossover methods all affect GA performance and must be chosen carefully to find the optimal solution in a reasonable amount of iterations.