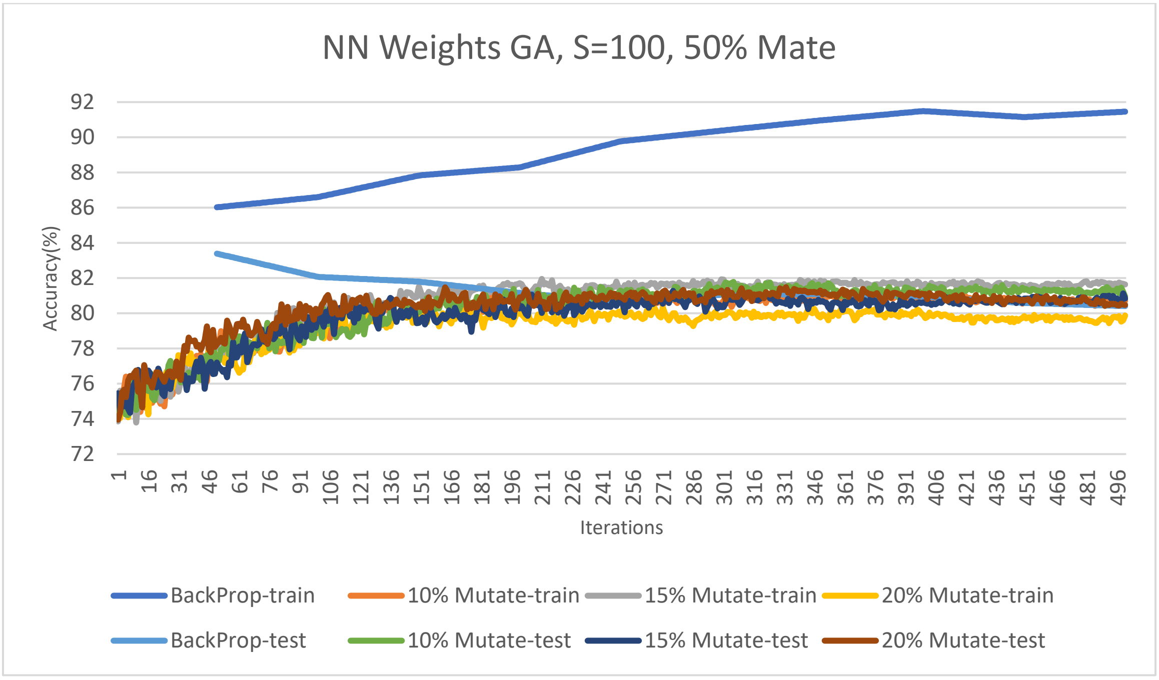
Neural Network Weight Optimization:

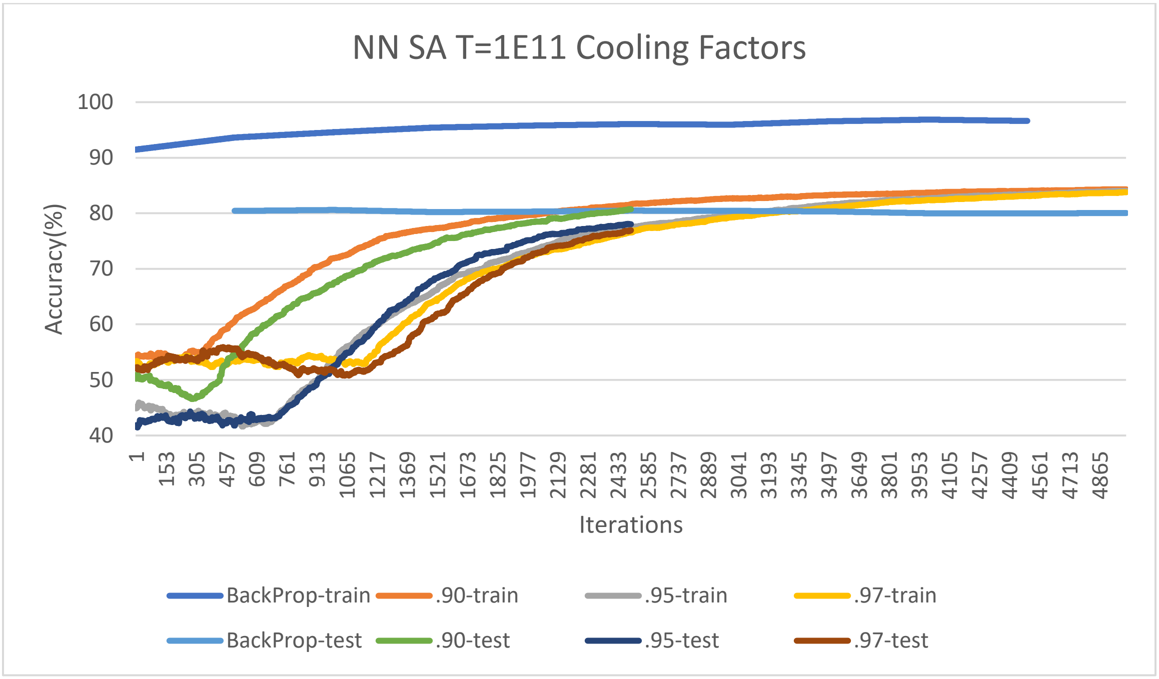
Due to the randomness of each algorithm, I ran 15 trials each of Simulated Annealing and Randomized Hill Climbing, and 5 trials of the Genetic Algorithm, and averaged the results. Also, due to the amount of time, I only trained on 10% of the training data (same as Assignment 1), and I only used 10% of the testing set as well. I also ran all the data through Weka filters: NominalToBinary, Normalize, and ReplaceMissingValues. For the Genetic Algorithm, I ran each variation for 500 iterations on both the training and testing sets. For Randomized Hill Climbing and Simulated Annealing, I ran each variation for 5000 iterations on the training set, and then 2500 iterations on the testing set. Every algorithm was run on a neural network with 54 hidden nodes, and the back propagation was run with a learning rate of 0.1 and a momentum of 0.1 found in assignment 1. Error bars were omitted on the iterations vs time graphs so as not to clutter them.



For GA, I used a population size of 100 in order to reduce the training time, no doubt a larger population size would produce better results. We can see the testing accuracy decreased from 10% mutation to 20% mutation suggesting 10% to be ideal, with an accuracy of 80.945% after 500 iterations. We can also see that GA performed well on the testing set compared with Back Propagation with 80.945% compared to 80.41%, however on the training set Back Propagation did much better with 91.46%, and GA only achieved 81.639% accuracy with 15% mutation. There is no sign of over fitting over 500 iterations as the testing and training accuracies both increase and seem to converge.

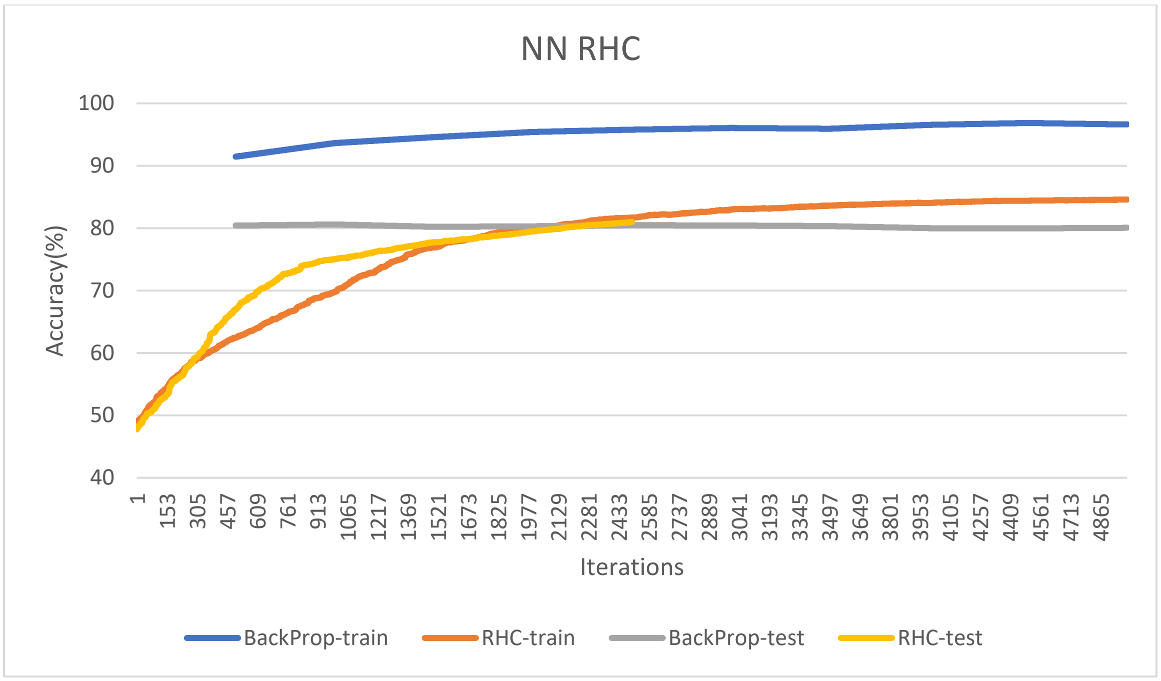
The mutation rate and mating rate allow the algorithm to deviate from the initial population, we can see that a higher mutation rate can make the algorithm converge faster. The graph shows the slight effects of tuning the mutation rate.

It took almost 30 minutes to run 500 iterations of GA on the training set, so 150 minutes for the 5 trials, which is more time than either SA or RHC took to run 10x the iterations. Each instance of the population needs to go through the update process, making this at least 100 times longer than SA and RHC. This actually seemed to take more time than backpropagation especially after the multiple trials, and it doesn't seem to produce better results, making it a poor choice for this problem.



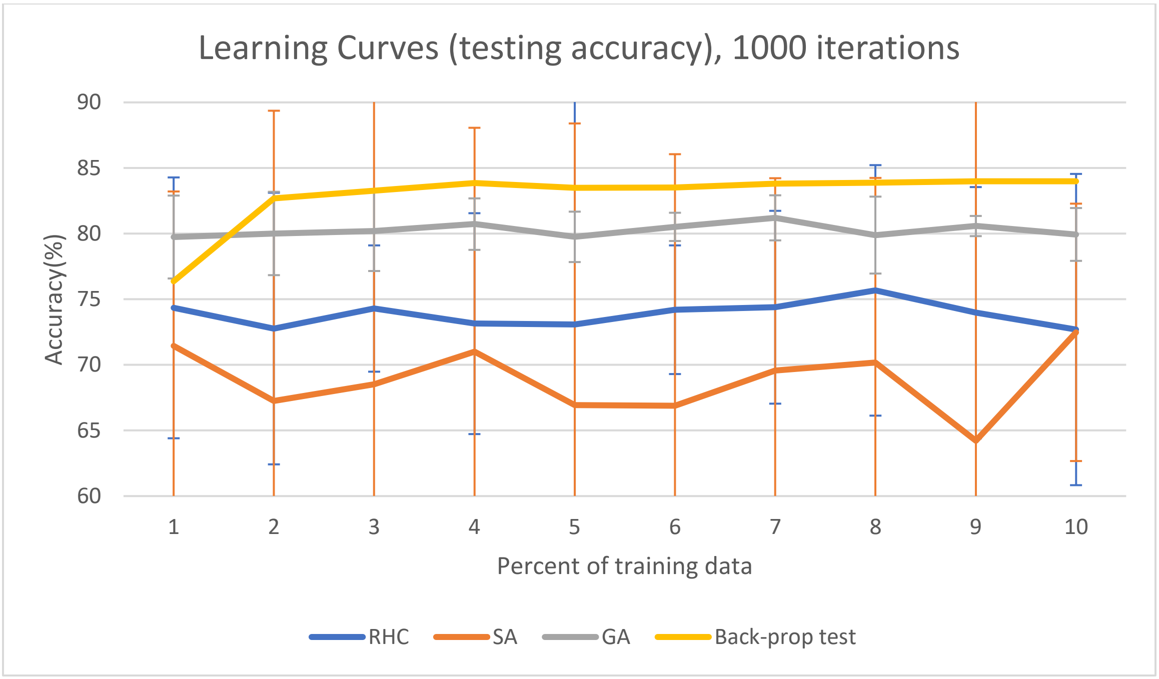
For Simulated Annealing, we can how the cooling factor changes the convergence time. With a larger cooling factor of 0.97, the "flat" period at the beginning is longer (lasts 1600 iterations), as SA is like a random walk when the temperature is really big. For 0.90 cooling, we see a "flat" section of about 300 iterations, which is much more ideal. There is no sign of overfitting on the first 2500 iterations since the testing and training accuracy both increase.

Simulated Annealing doesn't seem to perform any better than RHC, with 84.3% training accuracy (0.9 cooling) and RHC gets 84.6% training accuracy after 5000 iterations. The testing accuracies after 2500 iterations are also quite similar with SA at 80.7493% (0.9 cooling) and RHC at 80.985%. This suggests that the temperature probability is unnecessary and a hill climbing approach is more efficient. With more iterations, you could determine how quickly it take to reach back propagation training levels; however, considering the amount of time it take to perform 5000 iterations of this algorithm 15 times, about 3 minutes for each set of 5000. Overall, this takes less time than backpropagation(about 20 minutes for 5000 iterations), and produces similar results in terms of testing accuracy.



Randomized Hill Climbing performs quite well suggesting there are few local optima in the data. The testing Accuracy of RHC surpasses the Back Propagation testing accuracy at 2255 iterations. The increase in testing accuracy around 700 iterations is best explained by the imbalance of the data, or just it happened to reach a point that works for the testing set, and likely with more trials the curve would not be above the testing accuracy. The algorithm shows no sign of overfitting as the testing accuracy continues to increase over the 2500 iterations.

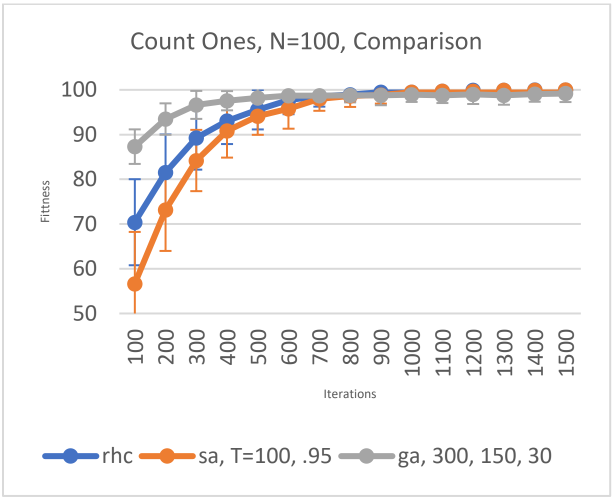
Computationally, RHC is very similar to SA but slightly simpler, taking only about 3 minutes for each trial of 5000 iterations and on a training set of 3000 instances. This is much less time than backpropagation takes (almost 20 minutes for 5000 iterations), and RHC is shown to have at least as good testing accuracy, making this a viable alternative to backpropagation for this data set.



For backpropagation the algorithm was run for 500 iterations, and is the same data from assignment 1, a learning rate of 0.1 and momentum 0.1 with 54 hidden nodes. RHC and SA were run for 1000 iterations with 15 trials each, while GA was run for 500 iterations and 5 trials. We can see the robustness of each algorithm with respect to the amount of data available for training from the graph above; especially GA, which barely changes at all from using 1% to 10% of the training data staying in the range of 79% to 82% testing accuracy.

We see that GA has much smaller confidence intervals, all smaller than +- 4%, than RHC and SA with have intervals up to size +-18% and +-26%, respectively. Since GA is much more consistent, it's clear that it is far more robust when it comes to the amount of available training data than either SA or RHC. Backpropagation also shows similar behavior with the only outlying value at 1% of the training data, otherwise it shows the same robustness to the amount of data.

Problem 1: Count Ones (Randomized Hill-Climbing):



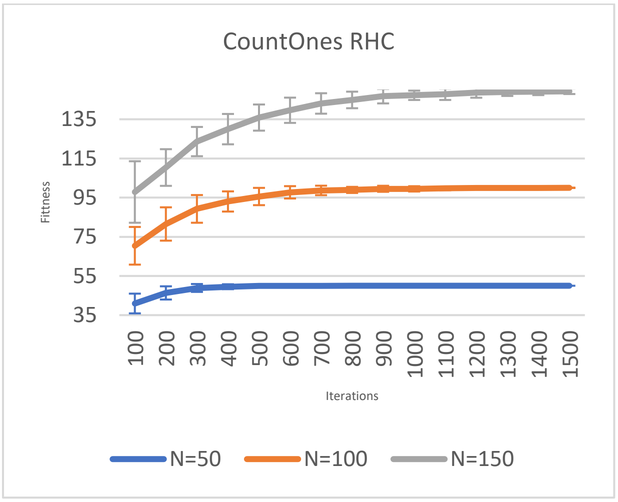
Count Ones is an optimization problem using a bit string of size n and an evaluation function which simply counts the number of 1's in the bit string. Clearly, the global maximum is at [1, 1, …, 1] and there are no other local maximums.

Each algorithm was run 30 times, and averaged. The error bars show +/- two standard deviations or a 95% confidence interval.

This is an ideal problem for Randomized Hill Climbing since there is only one local optima which is the global optimum, therefore there are no non-global optima for RHC to get stuck in. As we see in the graph RHC converges faster than either SA or GA to the global max of 100, reaching within (99, 100) in only 1200 iterations with 95% confidence.

Of course, Simulated Annealing does quite well, taking 1300 iterations to be within (99-100) with 95% confidence, which can be explained by the low temperature and low cooling factor. Since Simulated Annealing becomes Hill climbing at low temperatures, since the probability of going to a lower point is $e^{\frac{f(x^t)-f(x)}{T}}$ which goes to 0 as T goes to 0. There is absolutely no reason to use simulated annealing over RHC for this problem, as there are no local maxima so moving to a lower function value is always unideal.

A genetic algorithm also works quite well, and I found with a 300 population, a 50% mate rate, and 10% mutation rate, the algorithm will get within (97,100) within only 400 iterations; however, the confidence intervals never seem to get really small, as at 1500 iterations, the confidence interval is still +- 1.9, so there is much more variance than there is in RHC. Also, a genetic algorithm takes much longer than RHC and SA, and in a situation like this the extra time is not worth it.

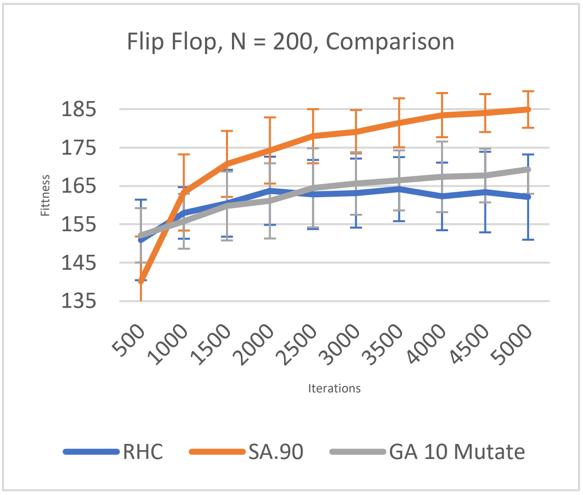


The above graph shows how the number of iterations before RHC converges to the global minimum increases with the size of the problem (size of the bit strings). For N = 50, we see that RHC will reach within (49, 50) after 500 iterations with 95% confidence. For N = 100, RHC will reach within (99, 100) after 1200 iterations with 95% confidence. And finally for N = 150, RHC will reach with (147, 150) after 1500 iterations with 95% confidence.

<u>Problem 2: Flip Flop (Simulated Annealing):</u>

For this problem, I ran each algorithm 30 times and averaged the results. The error bars show +/- two standard deviations or a 95% confidence interval.

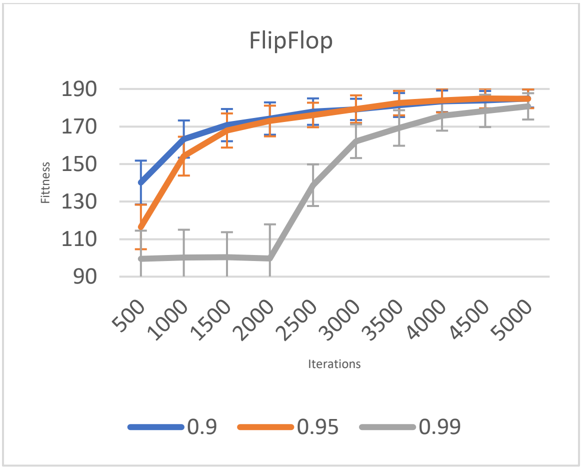
The evaluation function counts how many "flip flops" or changes are in the bit string (size N). For example, $[1, 0, 1, 0]$ outputs 3, while $[0, 1, 0, 0]$ outputs 2. So, clearly there are two global maxima at $[0, 1, 0, 1, \ldots]$ and $[1, 0, 1, 0, \ldots]$ and output value N-1. However, there are many local maxima for instance $[0, 1, 1, 0]$ outputs 2, but there is no way to change one bit to increase the output. This problem is interesting since there are many frequent local optima



Since there are so many local maxima, Simulated Annealing is an ideal choice, as unlike randomized hill climbing it has the ability to move in directions that don't increase the fitness function, and therefore doesn't get stuck in local maxima. This explains why Simulated Annealing is the only algorithm to reach anywhere near the global maximum of 199. I ran this algorithm with a cooling factor of 0.90 and a temperature of 1E10.

GA performs quite poorly (only 170/200), as cross over does little to increase the function value. Cross over at most adds another "flip flop" to the bit string increase the function by 1. This strategy is rather ineffective and unlikely to reach the global maximum. I ran this algorithm with a population of 100 with a 50% mate rate and 10% mutation rate.

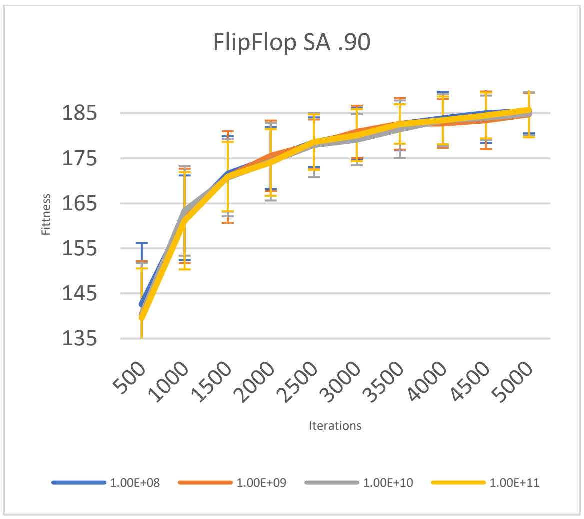
RHC is unlikely to ever reach the global maximum since there are so many local maximums. It only reaches a value of 163 after 5000 iterations, and the only way it can reach 200 if it precisely chooses every other bit to flip.



These variations were run with a temperature of 1E10, and a problem size of 200.

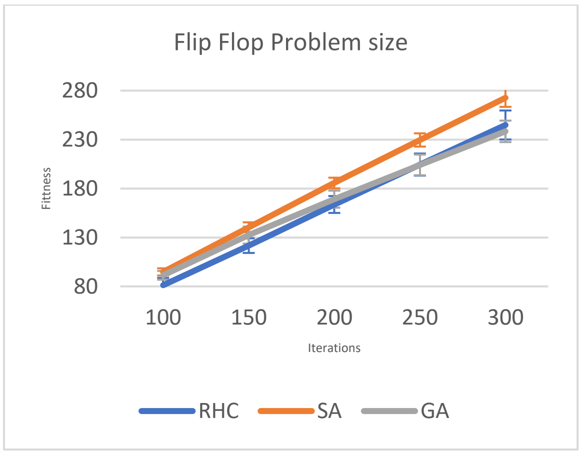
Here we see the cooling factors effect, on the convergence time of the algorithm. Clearly, 0.99 is too large as there is no upward trend until around 2000 iterations, since the temperature doesn't decrease quickly enough, so SA is like a random walk.

A Cooling factor of 0.90 yielded the best results with a function value of 184.9 +- 4.76 after 5000 iterations, only slightly outperforming the cooling factor of 0.95 which had a 184.76 +- 5 after 5000 iterations.



I ran the algorithm with different starting temperature with a cooling factor of 0.9 and a problem size of 200.

The starting temperature appeared to have little effect on the results with values all between 184 and 186 after 5000 iterations which is kind of expected at such high values and with such large numbers of iterations, since the temperature decreases to near zero long before 5000 iterations for each value.



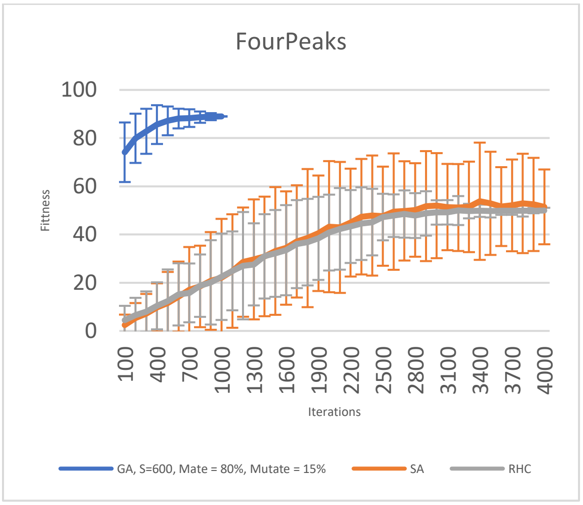
Each Algorithm ran for 5000 iterations. Simulated Annealing was run with a temperature of 1E10 and a cooling factor of 0.9. GA was run with a population of 100, a mating rate of 50% and a mutation rate of 10%.

We can see as the problem size increases, simulated annealing's advantage over the other algorithms also increase. GA progressively gets worse with the size of the problem, a higher population size would help fix this issue; however, a larger population increases the training time.

Problem 3: Four Peaks:

For this problem because of high variance, I ran each algorithm 200 times and averaged. The error bars show +/- two standard deviations or a 95% confidence interval.

This problem uses an evaluation function that takes the maximum of the number of consecutive 1's at the beginning of the string and the number of consecutive 0's at the end of the string, and then if both numbers are greater than a threshold ($\frac{N}{5}$) it adds N to the function value. Clearly, there are four local maximums: $[1, 1, \dots, 1]$, $[0, 0, \dots, 0]$, $\left[\underbrace{1, \dots, 1,}_{\frac{N}{5}+1} \ \underbrace{0, \dots, 0}_{N-\left(\frac{N}{5}-1\right)}\right]$ and $\left[\underbrace{1, \dots, 1,}_{N-\left(\frac{N}{5}-1\right)} \ \underbrace{0, \dots, 0}_{\frac{N}{5}+1}\right]$. This problem is interesting since it has a few sparse local optima.

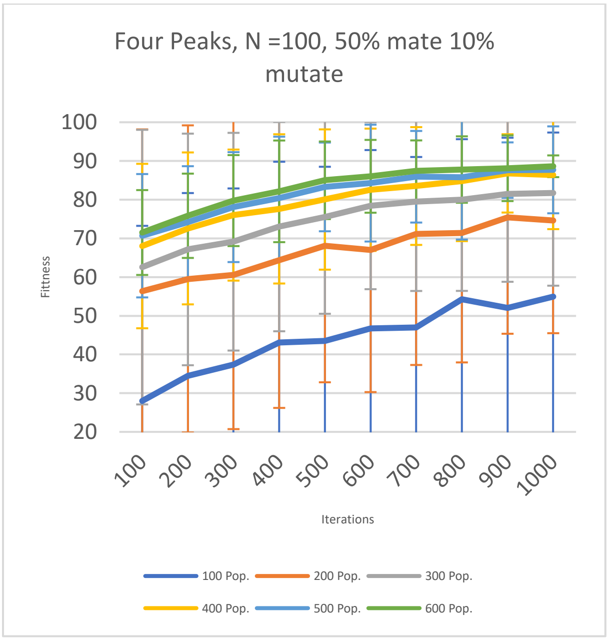


I ran the algorithms for N = 50, with a GA algorithm with tuned parameters, and SA with temperature 100 and a cooling factor of 0.95.

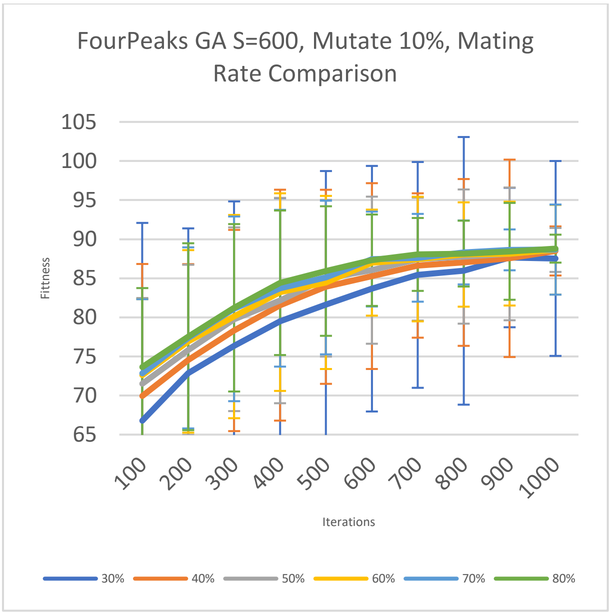
The Genetic Algorithm is the only algorithm that even get close to the global maximum of 89, neither SA or RHC get close. Intuitively, we can see why GA might perform well as if we look at the cross over of the two local maxima, $[1, 1, \dots, 1]$ and $[0, 0, \dots, 0]$. This produces a string $[1, 1, \dots, 1, 0, 0, \dots, 0]$, which over half the time is "above" the threshold and therefore close to the global maxima. GA also requires significantly less iterations to converge than RHC and SA, only taking 1000 iterations to reach confidence intervals of size +-2.8, whereas SA and RHC, take 4000 iterations and SA still has a large confidence interval, and RHC has a +-1.13 interval.

Simulated Annealing can occasionally find the global maxima (the final average was 51 at 4000 iterations), however the global optima are so far and sparse from the local optima that it is still unlikely to reach the global maxima.

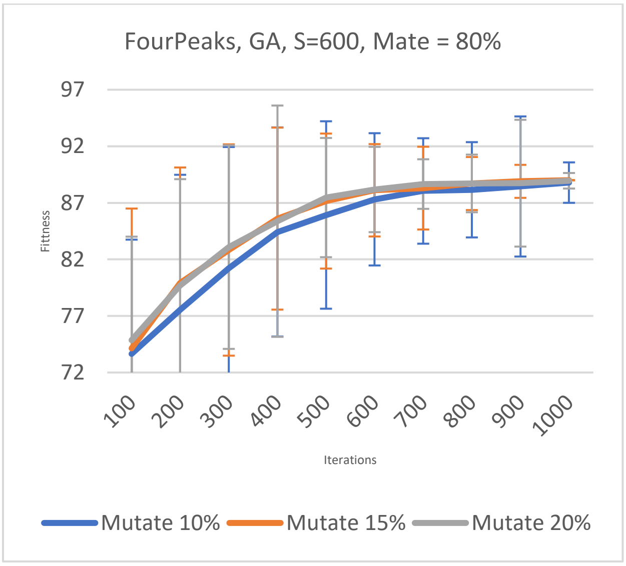
RHC has no chance of finding the global minima unless it starts "above" the threshold, which for N = 50, a probability of $\frac{2^{N-\frac{2N}{5}-2}}{2^N} = \frac{1}{2^{22}}$.



We see for a Genetic Algorithm, the population size of the algorithm increasing its fitness functions with the optimal value given time constraints chosen to be 600. A score of 88.625 +- 2.81 was achieved at 1000 iterations.

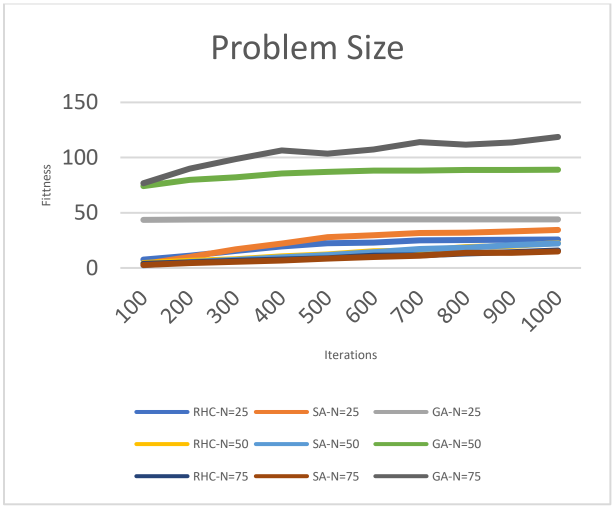


The Mating Rate seemed to have very little effect, especially after 1000 iterations. With scores ranging from 87 to 90; however, the confidence intervals shrank consistently from 30 % mating to 80%, implying more consistence between trials.



This was run with population size 600 and a mating rate of 80% and problem size 50.

The mutation rate from 10% to 20% seem to increase the rate of convergence. All three variations end near the same value of 88.5, however, 15% and 20% are consistently higher along the way. This makes since sine were increasing a rate of change basically, so the algorithm with the higher mutation rate (to some point) will change faster and hopefully converge faster. However, 15% mutation had smaller confidence intervals (actually reaching +- 0 at 1000 iterations), therefore 15% mutation is overall a more consistent model.



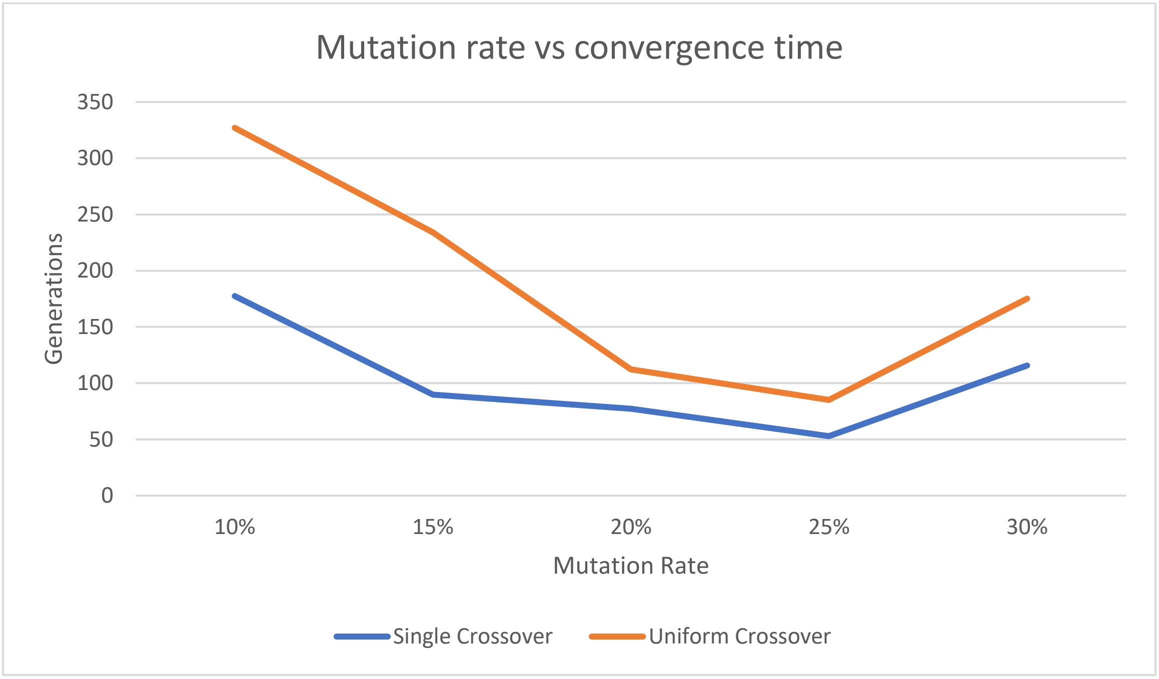
The GA model was run with a population of 600, a mating rate of 80% and a mutation rate of 15%. SA was run with a cooling factor of 0.90 and starting temperature of 100.

This graph shows how each algorithm perform for different problem sizes, exhibiting the robustness of a GA model as opposed to the RHC and SA models. We see that GA achieve after 1000 iterations the function values/global maximum: 44/44, 89/89, and 119/124 for N = 25, 50, and 75, respectively. However, SA and RHC have progressively worse values. RHC got 25/44, 22/89, 15/124, and SA got 34/44, 22/89, 15/124, but we do see that for small N, SA is much closer to the global max than for large N. SA is consistently “above the threshold” for N = 25 even though its average is not the global max. This makes sense since for small N SA has a higher probability to cross the threshold since it’s a larger percentage of the hypothesis space. And of course, we do see that GA is consistently above the other algorithms for each value of N; this makes sense given the high population size and the nature of crossover to help handle the large problem size.

Flappy Bird Problem:

For the purpose of this assignment, I decided that once a bird makes it passed the 10th pipe (13th pipe is created) then the model is sufficiently trained. I also ran each variation a total of 10 times. I also implemented another cross over function to compare to the one already being used. The default one simply swapped one weight between the parent models to produce the child model; I implemented uniform cross over which goes through each weight of the parent models and randomly decides whether to swap or not (code in flappy.py). Unfortunately, the variance over the multiply trials was extremely high (standard deviation of around 50 to 150); however due to time constraints, I was unable to perform more trials, in hopes of lowering the standard deviation.

The graph below shows the generations it took to sufficiently train models of different mutation rates and with different cross over functions.



For the single crossover (default), we can see it found a sufficient solution quickest for a 25% mutation rate only taking an average of 52.8 generations. The above graph makes intuitive sense, since for too low mutation rate the population will not change quickly enough and therefore will take longer to find an optimal solution; however, for too high mutation rates the algorithm becomes more random and thus will have higher variance and will take longer to find an optimal solution.

For uniform crossover, we saw again that an ideal mutation rate is around 25%, and took an average of 85.11 generations to become sufficiently trained. And again, the graph makes a lot of sense, as with too low a mutation rate it doesn't change quickly enough, and with a higher mutation rate the algorithm is too random. As for why single cross over did better than uniform cross over, I think this is because uniform crossover explores more possibilities (generally), essentially (most likely, since mutation) covering a larger region of the hypothesis space, and thus most of the time will take longer to find an optimal solution; however, the solution found might be superior, since there is more freedom in the way it searches.

Cross Over Function	Generation	Average Score
Single	248	1163.2
Uniform	39	273.6

I then ran each cross over variation at the ideal mutation percentage (25%) for 500 generations or until it reached the 100th pipe. Then used the trained models, and averaged the score obtained from 5 trials. Both trained models are included in the current model_pool folders.

We can see that it took the single cross over longer to converge to a solution that passed the 100th pipe, 243 generations, while it only took the uniform cross over 39 generations; however, the single cross over has a much higher average score of 1163.2 to 273.6, and in the model, we can see that many more birds make it passed more pipes than in the uniform cross over model which only has a couple birds that make it passed even the 20th pipe. Also, the 39 generation is much lower than the average that it took to even reach pipe 10, so the algorithm was lucky so the Generation number isn't particularly meaningful. Although this looks inconsistent with the previous data, I think as I increase the threshold of sufficiently trained, the uniform crossover function is more likely to converge faster than the single crossover since it (most likely) explores more significantly different hypotheses in fewer generations, and therefore the target function is more likely to be found without extensive amounts of mutation by the uniform crossover.