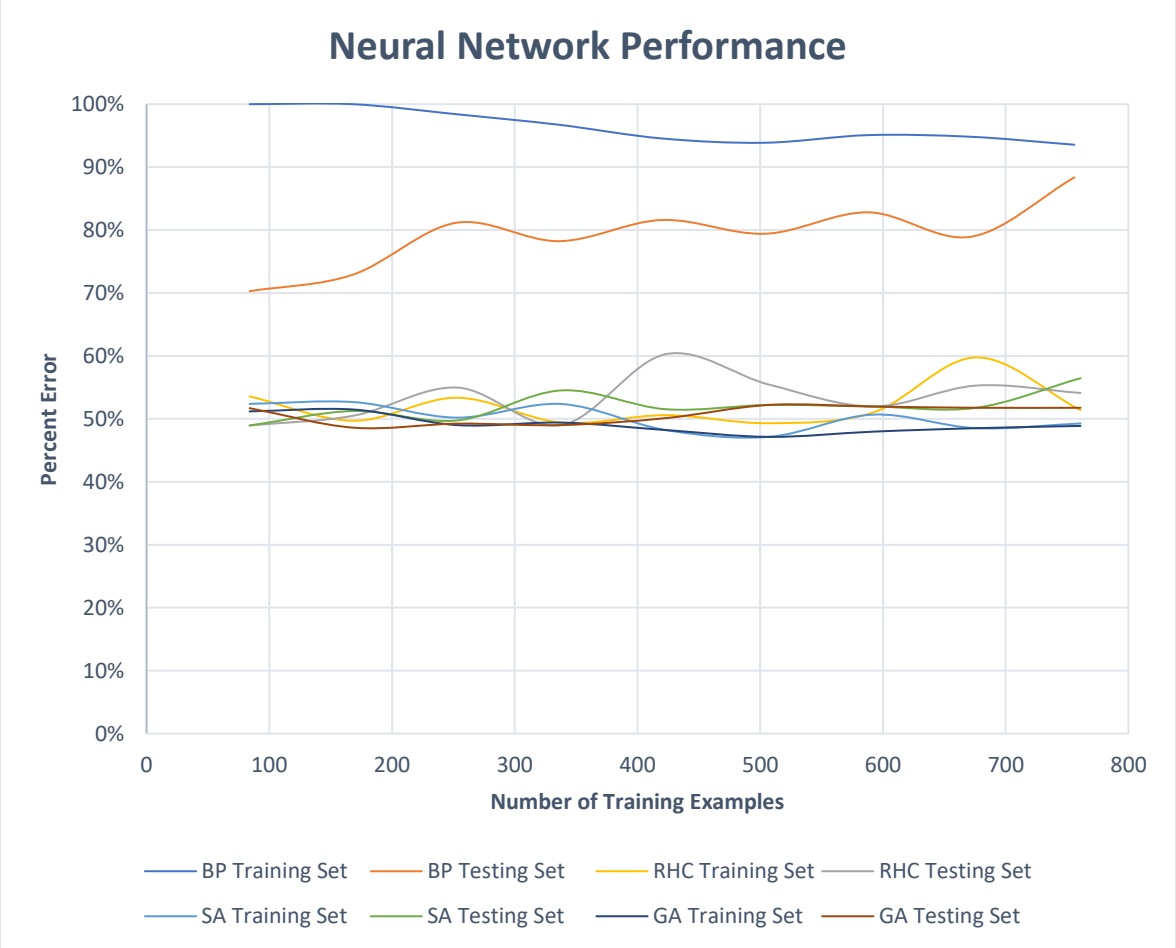
Jackson Cook

CS4641 – Machine Learning

Assignment 2: Randomized Optimization

The Problems Given to You

I used the Vehicle dataset. This dataset was collected with the goal of classifying a vehicle by its silhouette as one of four types. By extracting a variety of scale independent features, the vehicle would be classified as OPEL, SAAB, BUS, or VAN. Two sets of 60 images were taken for each vehicle covering a full 360 degrees of rotation. This dataset contains 846 instances and each has 19 attributes.



Graph NN: Shows training and testing accuracy of each method of error changing with each algorithm with varying quantities of training examples

Back propagation performed significantly better than Simulated Annealing, Randomized Hill Climbing, and the Genetic Algorithm. I presume this is due to the space being too large for the neighborhood algorithms (Simulated Annealing and Randomized Hill Climbing) to be effective, as well as how the space is filled with many

similar cases (local maxima). I also believe the Genetic Algorithm would have performed better with more iterations but would have taken too much time, already taking many minutes to run with the standard number of iterations. The randomized optimizers performed twice as well as random chance (25%) on average which shows they are not useless.

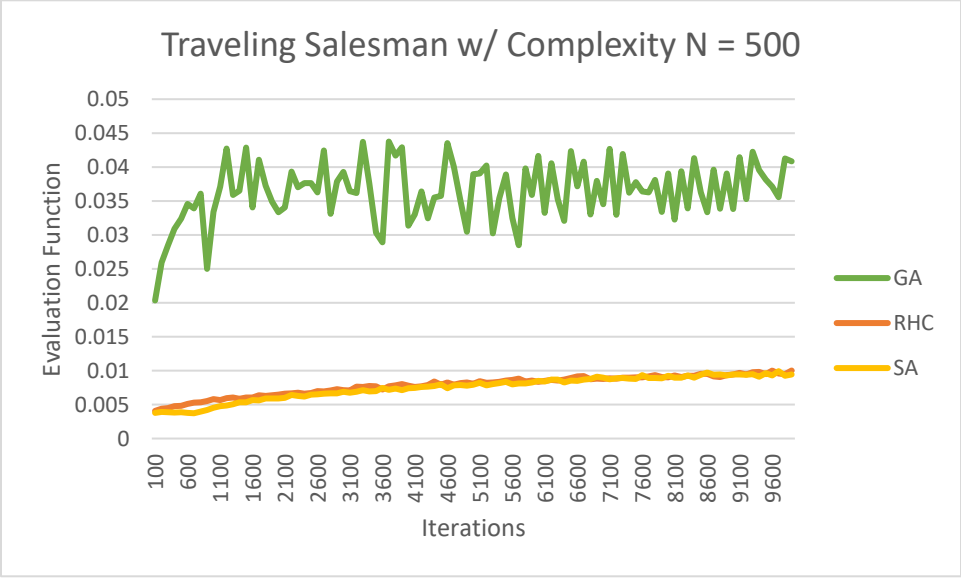
I ran each of the randomized optimization algorithms with varying numbers of iterations and it did not have any effect on their accuracy and showed no new results, so I have chosen not to include that graph. I also tried changing hyper parameters for the Genetic Algorithm (mutation rate, mating numbers, population) and Simulated Annealing (initial temperature, cooling rate) with constant iterations, testing set, and training set, but no changes in the accuracies were seen.

The Problems You Give Us

In collection of this data, 5 iterations were run over each collection and were averaged. This was to mitigate randomness from the collection process.

i) Genetic Algorithm:

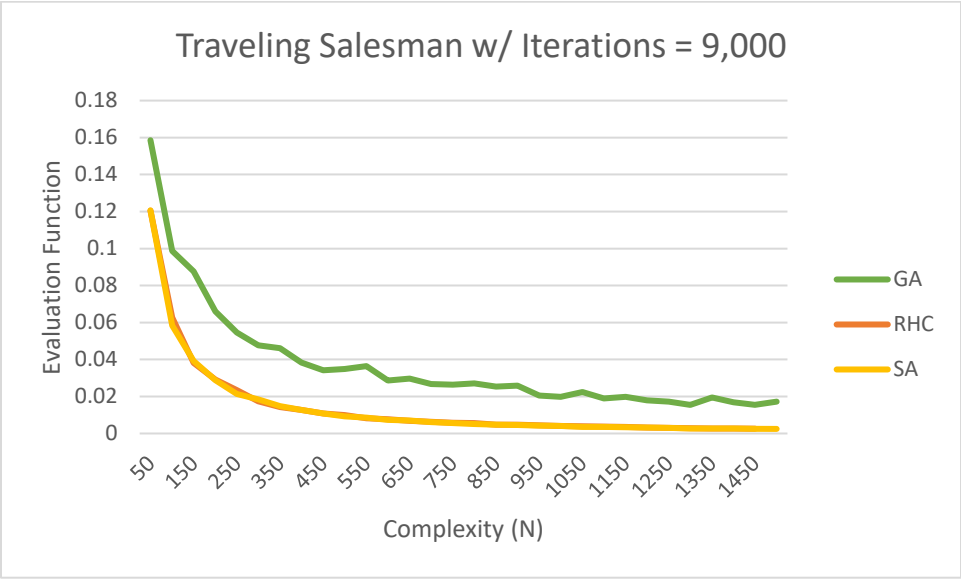
To highlight the power of Genetic Algorithms, I chose the **Traveling Salesman Test**. This test gives the algorithm N number of distances to neighboring cities and it is the algorithm's job to determine the shortest way to visit all of the cities.



Graph GA1: Shows performances of all algorithms given complexity N = 500 over many iterations (100 through 9600)

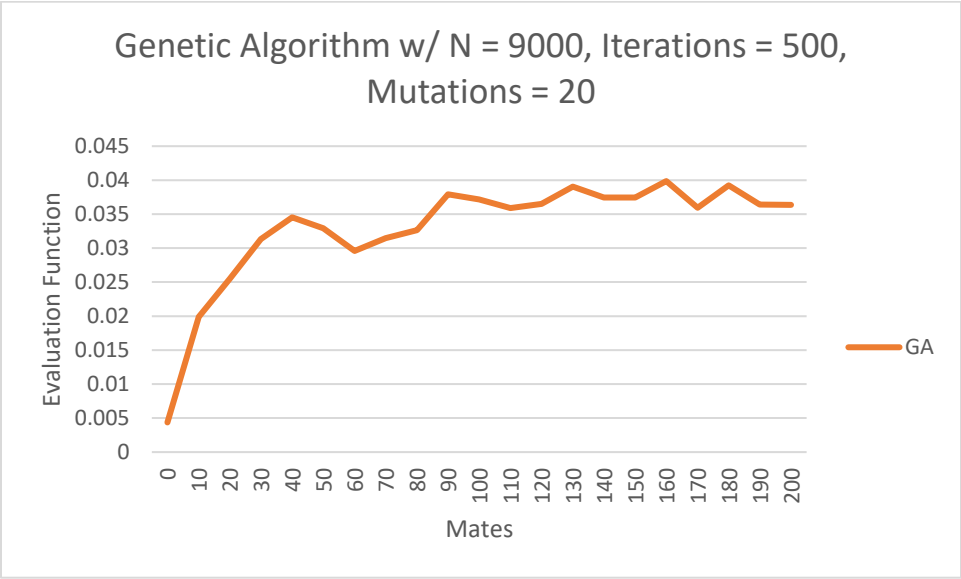
The Genetic Algorithm leaves the other algorithms in the dust on this test. This is likely due to the space having an unimaginable number of local maxima to trap the Randomized Hill Climbing and Simulated Annealing algorithms. The Genetic

Algorithm is able to pinpoint the global maxima with its strong global search, yet was unable to converge completely, oscillating around 0.038.



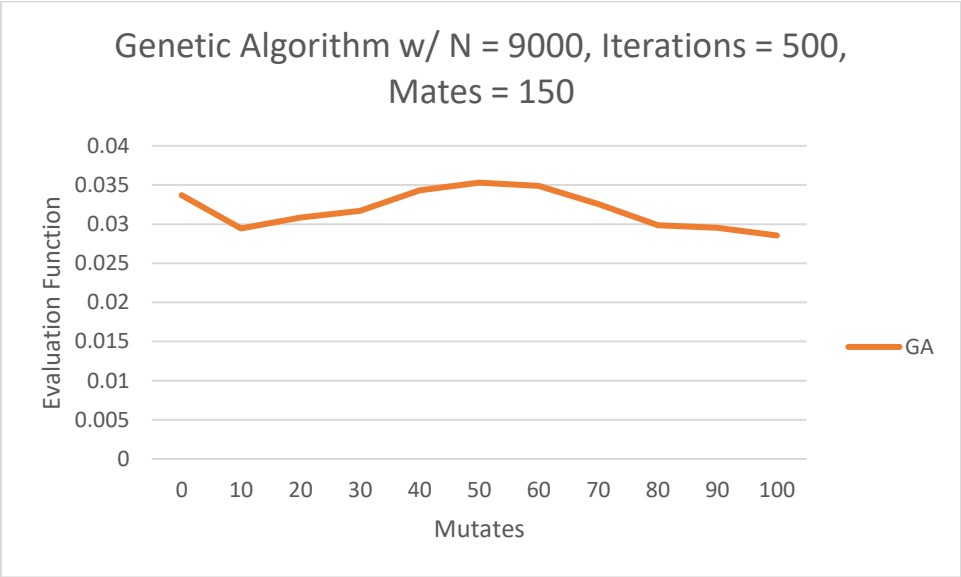
Graph GA2: Shows performance of all of the algorithms with constant Iterations = 9,000 with varying complexity N

The Genetic Algorithm continued to outperform the algorithms as the complexity increased. The gap between Generic Algorithms and the others did seem to shrink however. This is likely due to the constant iterations I used as opposed to increasing the iterations with the complexity since the Generic Algorithm can take many iterations to perform well due to its global search.



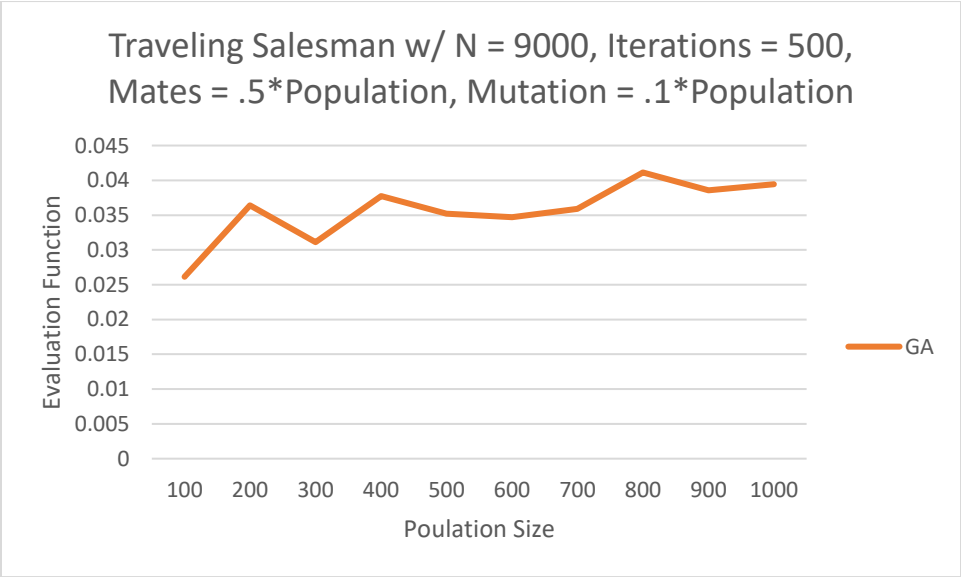
Graph GA3: Shows a Genetic Algorithm with varying numbers of mates

The effectiveness of the algorithm seems to plateau at around 100 mates and stays steady to 200 mates. This keeps the population more uniform combining strong competitors with others allowing the algorithm to perform well.



Graph GA4: Shows a Genetic Algorithm with varying numbers of mutated members

The above graph shows the effectiveness of the Genetic Algorithm with varying numbers of mutated members. There seems to be a peak at 50 with a rather steady incline beforehand and a decline afterwards. This points towards there being an advantage to have more than 0 mutated members in the algorithm, but it also says that there is a point where there are too many mutated members, causing the algorithm to be distracted and moving it farther from the maximum.

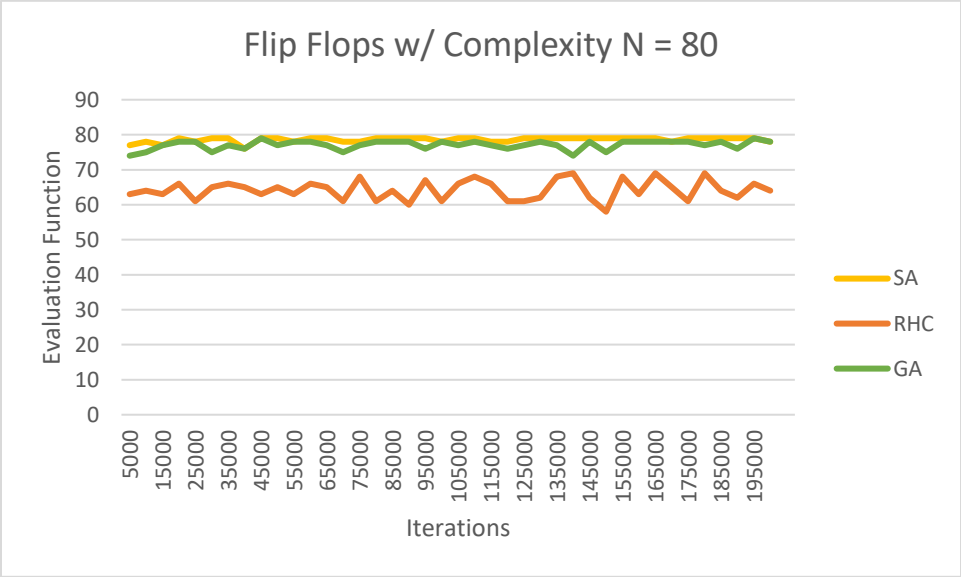


Graph GA5: Shows a Genetic Algorithm with varying population size but constant proportion of mates and mutation

Increasing the population created oscillation, but there seemed to be an increase in the Genetic Algorithm's performance on the test. This is due to there being more learners to grow the search for the global maxima.

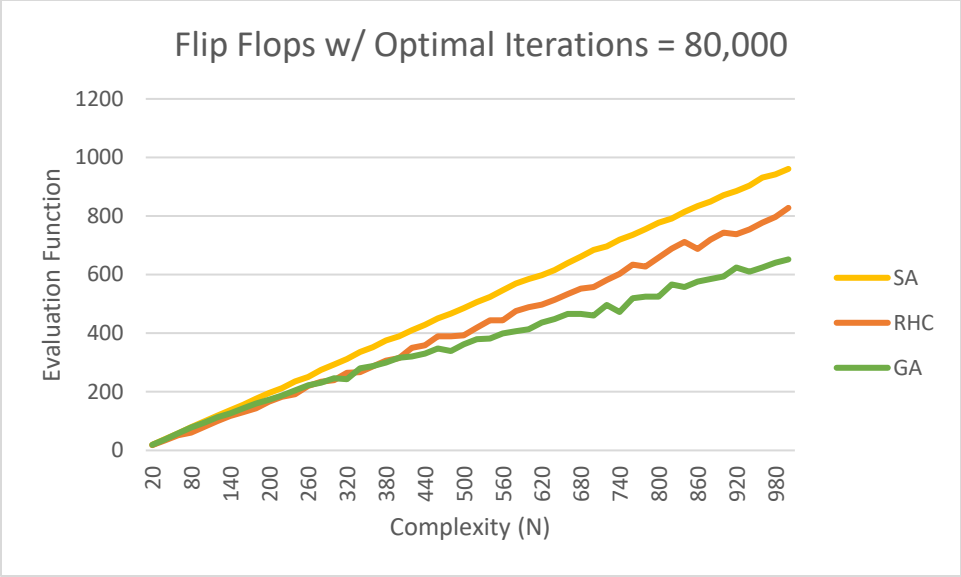
ii) Simulating Annealing:

To highlight the power of Simulating Annealing, I chose the **Flip Flop Test**. This test gives the algorithm an array of (N – 1) 1's and a single 0. The algorithm is to then maximize the number of bits that are next to the "flipped" (0).



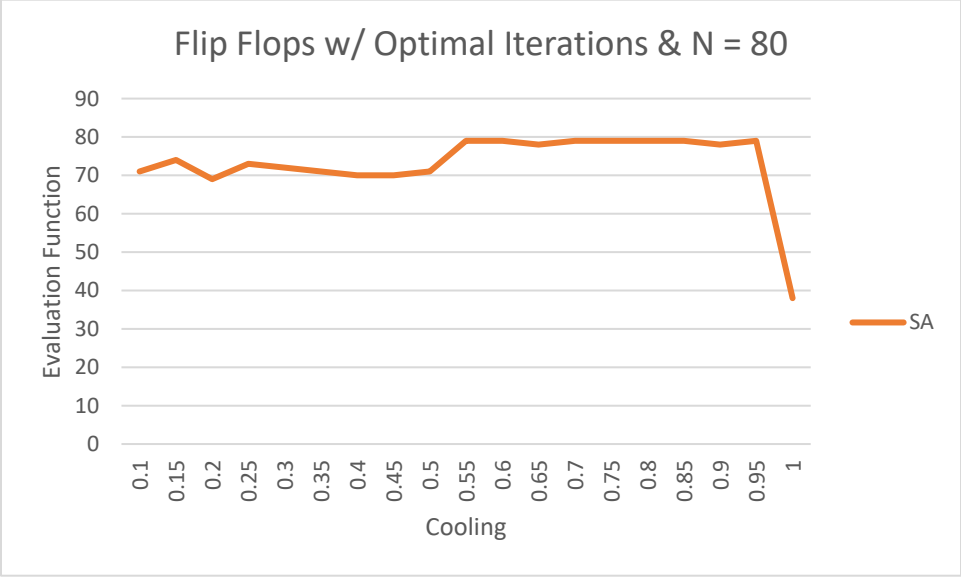
Graph SA1: Shows performance of all of the algorithms with N = 80 and varying iterations (5,000 through 195,000)

As can be seen in the graph above, Simulated Annealing performs better than the other algorithms consistently, but Generic Algorithms was a close second. Simulated Annealing performed better (converging on 79 at around 20,000 iterations) on this algorithm due to its ability to turn around and go the other direction. For example, in some instances if it were to continue one way, it may go the opposite way of the 0 making it unlikely to fall into local maxima. This is why it outperformed Randomized Hill Climbing which would continue climbing one way but may be going the opposite direction it needs to (getting caught in local maxima). The Generic Algorithm never really converged, but oscillated around 77 and 78 starting at 45000 iterations. Generic Algorithms had more oscillation due to its disposition to randomly mutate and/or mate, but it performed well because it is made to quickly find the global maximum.



Graph SA2: Shows performance of all algorithms with varying input size/complexity N with constant iterations 80,000

As the complexity/size of input grew, Simulated Annealing really started to show its power in tackling this problem. Generic Algorithms performed much better at lower N, but struggled as N grew. This could likely be solved, but generic algorithms are already very computationally heavy. Simulated Annealing and Random Hill Climbing for the most part continued fairly linearly as 80,000 iterations was plenty for those algorithms to perform well even as N skyrocketed.



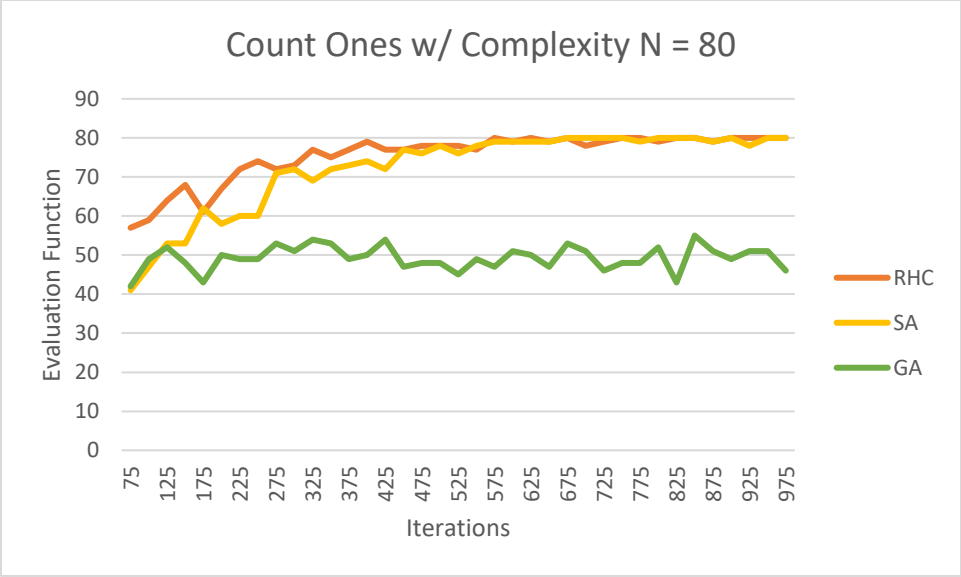
Graph SA3: Shows performance of Simulated Annealing w/ constant iterations & complexity w/ varying cooling

This graph illustrates how changing the cooling parameter of Simulated Annealing effects the performance of the algorithm. The optimal value of cooling for this situation seems to be 0.6 to 0.95. I noticed a large drop in performance

after cooling went over 1 so I did not include that in the graph besides 1 itself. This indicates that a cooling of greater than 1 decreases the temperature too quickly for the algorithm to find a good maxima and got it stuck in local maxima. I also ran evaluations on the starting temperature of the algorithm, but the starting temperature seemed to not affect the performance of the algorithm at all so I did not include that graph.

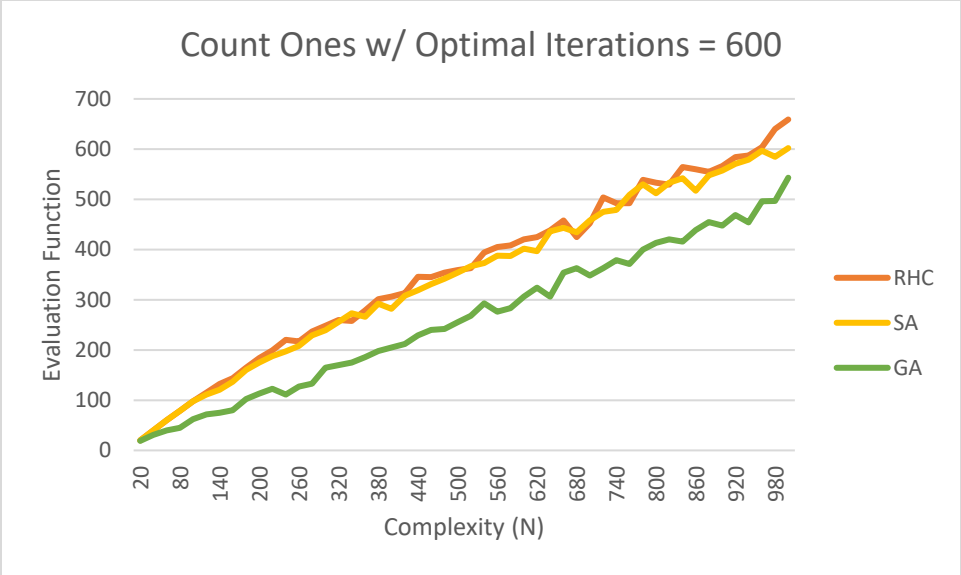
iii) Randomized Optimization:

To highlight the power of Randomized optimization, I chose the **Count Ones Test**. This test challenges the algorithm to determine the number of ones in an array of size N filled with 0's and 1's.



Graph RHC1: Shows performance of algorithms with complexity N = 80 over varying iterations (75 through 975)

Randomized Hill Climbing performed the best on this algorithm. It converged at around 400 iterations, about 50 iterations before Simulated Annealing converged (both on 80). This is due to Count Ones being a very linear problem. Randomized Hill Climbing would continue moving towards the single maxima which is both local and global and would reach it very quickly. Simulated Annealing took slightly longer due to its tendency to move the wrong way when there actually was only one maxima in the space. The Genetic Algorithm struggled with this test, oscillating and never converging, staying around 50 on the evaluation function. Since the Genetic Algorithm searches on a global scope, it would need many more iterations to do well on a simple test like this.



Graph RHC2: Shows performance of algorithms with Iterations 600 with varying complexity N (20 through 980)

As the size of the space increased, all of the algorithms continued to perform the same on the test, maintaining a nearly linear slope, but started to oscillate more as the complexity increased due to the new variance the difficulty introduced. The increase in the complexity did not shed any new light on the performance of the algorithms otherwise.

Flappy Bird

There were many modifications that could be made in the flappy bird problem:

Mutation Rate Modification

flappyV1.py (Mutation rate = 90%, everything else as given)

Increasing the mutation rate while keeping all other aspects at the given constants caused the birds to be extremely volatile in learning (15% to 90%). Sometimes after as few as 25 generations the score could reach over 100 for the birds but more often than not the birds would finally break through and get a score of 1, then the next iteration would go back to failing (low consistency). I believe this is due to the volatility that is introduced by the randomness added by the frequent mutations, causing previous mutations that may have been useful to be overwritten by new less useful mutations. When I tried extremely low mutation rates (5%), the took a very long time to separate and eventually begin to start navigating the pipes (>>30). The score for these birds was often made by one or two birds, not a large group of birds.

Mutation Rate Modification + No Cross-Over

flappyV3.py (Mutation rate = 90%, Cross-Over removed

This combination resulting in slower convergence, but once the convergence was reached the birds were more steady with their performance. The implications of this code would mean the best birds are chosen, but they are not bred with the other best birds, rather they are just passed on to the next generation as they were before, leaving mutation as the only aspect of the algorithm providing variance. When convergence is reached shortly after 50 iterations, it is normally only one bird that is performing well (scoring over 20).

Mutation Magnitude Modification

flappyV2.py (Mutation magnitude .5 -> 1, everything else as given

To the contrary, when the frequency of mutation was maintained but the influence of the mutation (± 0.5 to ± 1) was increased (the size of change of weight from a mutation), the birds consistently reach well over 20 score after any number of generations between 20 and 50. These birds stayed in groups more as they are more similar genetically. I believe keeping the mutation rate at a low 15%), but increasing the influence caused the mutations that did occur to be more impactful while not infecting the whole population that may already be well on its way to being successful. This kept the birds consistent, so few drops were seen once convergence was reached. Also, I believe the small mutation rate is what made the large number of iterations necessary. This is due to the evolution of the birds happening slower.

Conclusion

Randomized Hill Climbing excelled on problems that had only one maxima in the space. RHC tends to get caught up in its constant ascent to the top of the hill without ever considering going down the hill or past the hill. Its ability to randomly restart does not fend off this tendency perfectly. For the Count Ones Test, Randomized Hill Climbing excelled due to its constant travel upwards on the singular maxima in the space. The main problem with Randomized Hill Climbing was the struggle it faced with more complex problems with more than one solution/maxima.

To contrast, Simulated Annealing performed better on problems with many local maxima but also one global maxima nearby. When the cooling rate is at an appropriate rate for the SA, the algorithm is able to hop between maxima to eventually find a global maxima if it is nearby. This is accentuated in the Flip Flops Test. Randomized Hill Climbing tended to struggle in this space due to its many local maxima. The main problem with Simulated Annealing was that it could still get caught in local maxima in large spaces.

Generic Algorithms were the only of these that could tackle problems that had huge spaces with many local maxima but also one global maxima. This is due to the robustness of the algorithm and its many ways to explore other parts of the space (mutation, mating, population size). This of course is highlighted in the Travelling Salesman Test where GA far outperformed the other algorithms by exploring the whole space and not getting trapped in local maxima. The main problem of Generic Algorithms was the time it took to run.