Randomized Search Algorithms – An analysis on Random Hill Climbing, Simulated Annealing, and Genetic Algorithms

1 – Comparison of Randomized Optimization Algorithms on weights of a Neural Network

1.0 – Intro

I once again used the UCI Breast Cancer dataset. The training and testing split utilized was 70/30. Times to complete the algorithms are included in each section, and they all scale linearly with the number of iterations, which makes sense. Training and testing time were the same which is why there are not separate data. Interestingly, the testing set outperformed the training set for all 3 algorithms. I can attribute this to the way the splitting was performed. In the previous HW, I was able to create randomized splits of the data. It is likely that this dataset has sections of non-independent data, and that the test split contains data which lacks noise and are very similar to each other.

1.1 - Randomized Hill Climbing

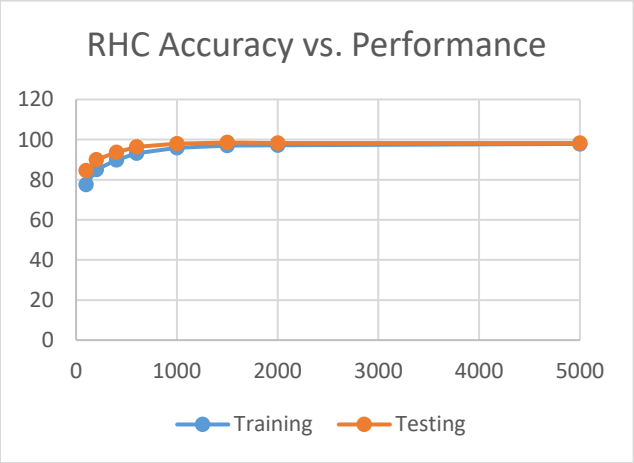

Figure 1.1a - RHC Accuracy (% of correctly predicted labels) vs number of iterations

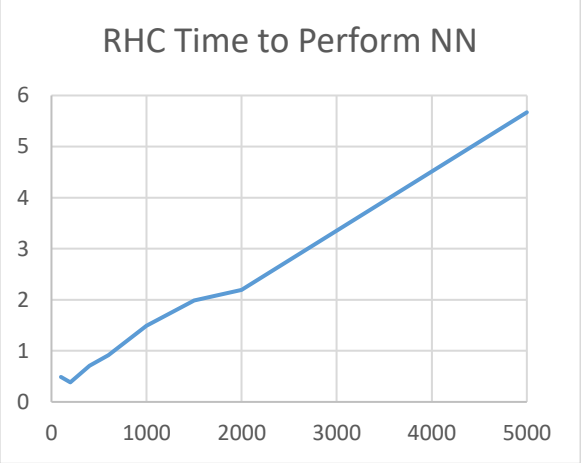

Figure 1.1b – Time to complete RHC vs number of iterations

RHC performs well in functions with no underlying structure, and does very well in pure optimization problems. However, in this case we are trying to "optimize" our continuous, real-valued neural network weights. In the context of this problem, it is likely that function of the weights does not have many local minima, which is why RHC performs so quickly and accurately.

RHC converged very quickly and was the best performer out of all the algorithms. This is potentially because the optimum weights for this particular dataset have a wide basin of attraction and thus the algorithm does not get trapped in any local minima and reaches the global optima immediately. It would be difficult to "improve" upon RHC since it really doesn't have any parameters - there's no tuning except increasing the number of iterations. In that sense it is a good baseline judgement for the remainder of the algorithms.

1.2 – Simulated Annealing

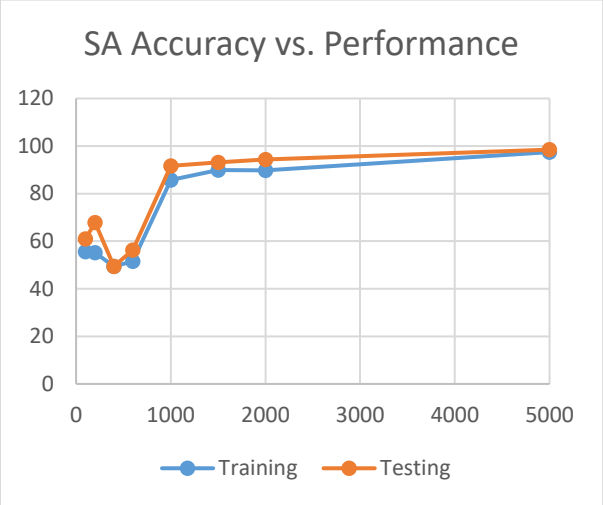

Figure 1.2a - SA Accuracy (% of correctly predicted labels) vs number of iterations

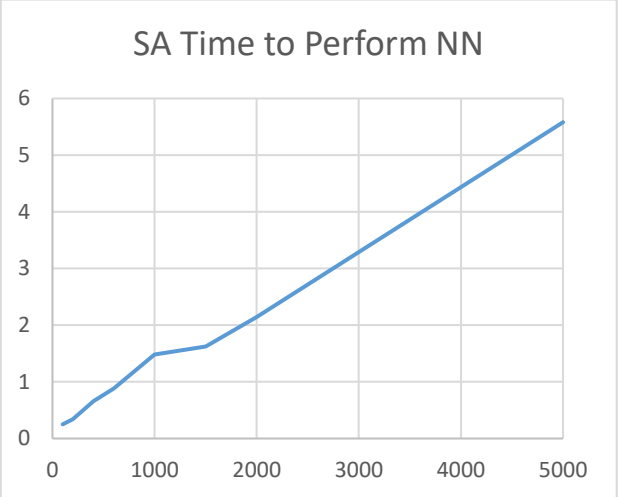

Figure 1.2b – Time to complete SA vs number of iterations

Simulated Annealing starts out very low to begin, and even *decreases* initially. The temperature parameter starts at 1E11, with a cooling rate of .95. With such a high starting temperature, SA essentially initiates a random walk, traversing or exploring the data to get out of local optima. It has plenty of time to do this with a cooling parameter of .95.

As you can see from figure 1.2a, from iterations 0-500 SA explores the data and gets stuck in local minima (which explains the sharp decrease in accuracy initially). However, after about the 500th iteration, SA has likely found a global maximum value as it increases sharply in only 500 more iterations, eventually converging after 1000 iterations. The graph quite literally portrays the SA algorithm climbing a likely global optima.

SA performs as quickly as RHC and is just under it in terms of accuracy. It is likely that SA is cooled into high iterations and thus does not explore any more. Thus there is a possibility that SA is stuck in a local optima just below RHC in the high iterations of the algorithm.

1.3 – Genetic Algorithm

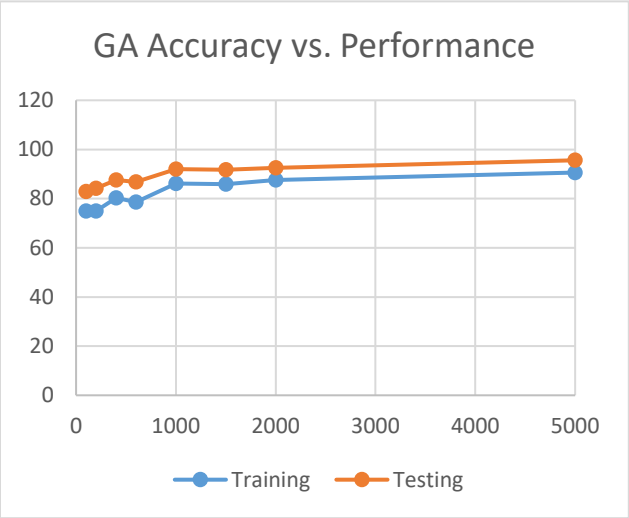

Figure 1.3a - GA Accuracy (% of correctly predicted labels) vs number of iterations

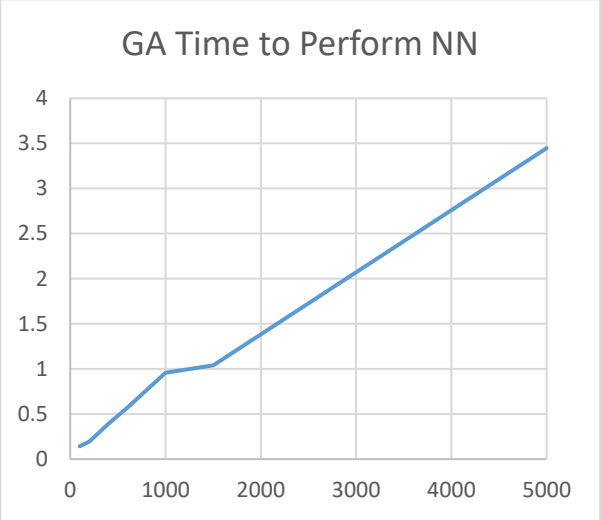

Figure 1.3b – Time to complete GA vs number of iterations

This Genetic Algorithm performed the worst out of all the 3 algorithms. It starts with a somewhat low performance and plateaus rather quickly. Parameters were set to population size of 50, 20 mates, and 10 mutations per generation. In this particular type of problem, it would make sense for a GA to perform the best. The structure of the weight function for a NN is likely complicated more structured which would benefit a GA nicely, as the mutations and crossover function would allow it to learn the underlying structure and conform to it in a manner that RHC and SA could not.

Increasing the population and hence the mates and mutations would likely increase performance for the GA. It would aid the algorithm in learning the structure, but would require more iterations to find an adequate population. However, I can only assume that the GA in this case is preventing overfitting. The mutations and crossover provided a more "natural" balance to the NN which allows it generalize better to the dataset. Forcing an increase in accuracy would help bump the performance perhaps a few percentage points, but it would still see the same flat lined structure that it currently has.

2 - Optimization Problems

2.1 - Knapsack Problem

The Knapsack Problem is defined as follows (as per Wikipedia):

"Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible."

Abigail uses a bounded knapsack problem and sets the upper limit on number of copies per items to 4. Essentially,

$$\text{maximize} \sum_{i=1}^{n} v_i x_i$$

$$\text{subject to} \sum_{i=1}^{n} w_i x_i \leq W \text{ and } 0 \leq x_i \leq c$$

where Wi are the weights of the items, Vi are the values, Xi are the number of copies, and W is the max weight.

Figures 2.1a and 2.1b are representative of iterations vs. maximum weight values and complexity (in this case is the number of items in the knapsack for N = {20,40,……160}) vs accuracy, while Table 2.1c displays convergence time vs. # of iterations.

GA has the best performance for this particular problem. It heavily exceeds RHC and SA over 1000 iterations. The nature of this problem allows for GA to excel, especially because it is an unbounded knapsack problem. This type of complex and diverse data space makes the GA a great performer in this instance.

We can notice that initially GA appears to gain higher scores as we reach a higher complexity in our fitness functions (more items), however it eventually converges with RHC and SA. This is likely because the complexity of the function increases the amount of local minima, and the problem becomes more structured with a larger, diverse distribution. This allows the GA to outperform RHC and SA because it learns the structure over many generations. However, the convergence becomes apparent after around 140 items. The complexity has increased and GA is likely caught in many local minima with only 1000 iterations. The GA would likely perform better once again if the number of iterations is increased to match the complexity.

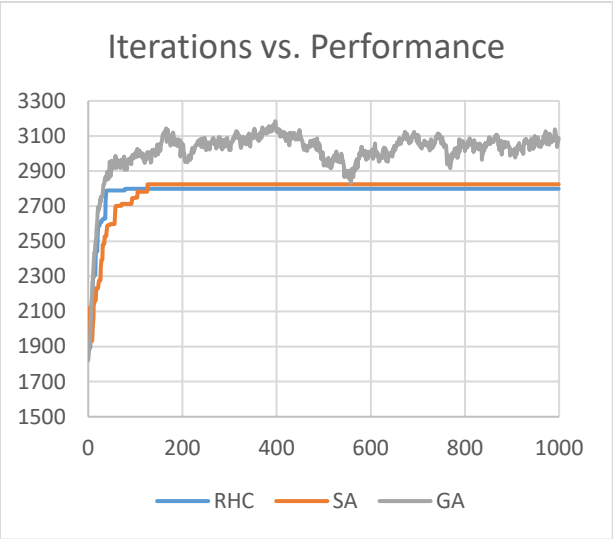

Figure 2.1a – Number of Iterations vs Weight of Items
(Items in Knapsack = 40)

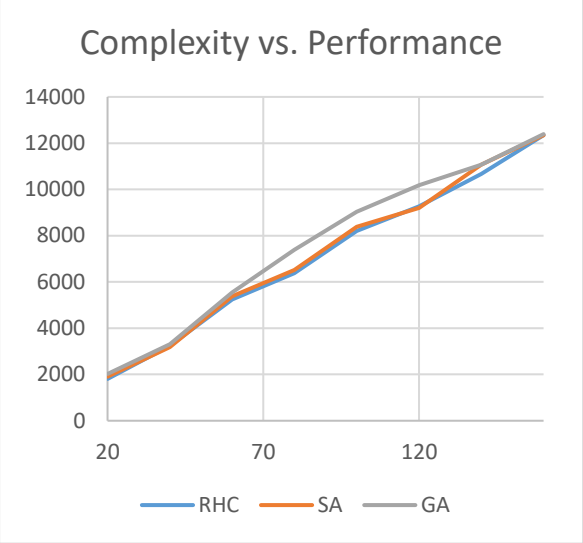

Figure 2.1b – Complexity (Size of Knapsack) vs Max Weight Obtained (Iterations=1000)

Table 2.1c – Knapsack Convergence Time

At 150 Iterations	
RHC	14 ms
SA	9 ms
GA	176 ms

Finally, noting Table 2.1c, as is standard GA had the slowest performance time. This is understandable as it must compute a collective "pool" of knapsacks and their subsequent generations during each run.

2.2 – Count Ones Problem

The count ones optimization problem aims to maximize the number of '1''s in a given bit string of size N. It is a very simple problem as it incentives adding more 1's to the bit string. Figure 2.2a illustrates iterations vs performance at N=150 up to 1000 iterations. Both RHC and SA converged to an optimal solution at around 500 iterations, while GA converged very quickly at a suboptimal level. Complexity (in this case, the size of the bit string) vs performance was measure at values of (N = 150,200,250,300,350,400,450,500).

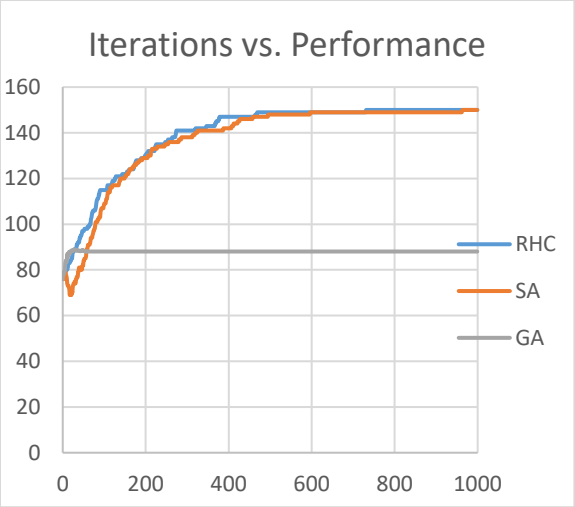

Figure 2.2a – Number of Iterations vs Maximum Count of 1's Obtained (Size of Bit String=150)

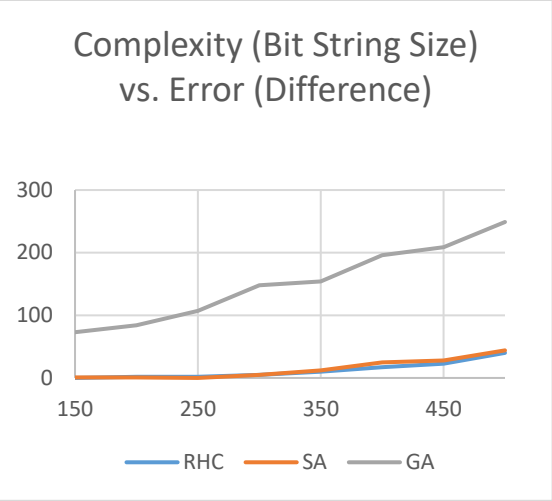

Figure 2.2b – Complexity (Bit String Size) vs Error (N – max value given by algorithm), (Iterations=1000)

RHC performs the best (and much better than the GA) because the count ones problem is a pure optimization problem – it aims to maximize the number of ones in the bit string. The set is comprised of *independent* 0's and 1's, thus the space is not diverse and does not contain an underlying structure. This deters the performance of the GA because it benefits from diverse and structured functions where it can make leaps out of local minima and into another part of the large and diverse space. This also explains why it completely plateaus – its mutation function does not provide it any benefit and it fails to gain from the non-existing structure of the data.

Table 2.2c – Count Ones Convergence Time, N=40

At 500 iterations:	
RHC	1 ms
SA	34ms
GA	120ms

Additionally, we can see how the complexity of the problem gives GA a much harder time than SA and RHC. As complexity increases, the GA's error accelerates while RHC and SA appear to very slightly increase. I would assume this is because of the nature of this optimization problem – for hill climbing algorithms like RHC accuracy scales linearly (or slightly increasing) with number of iterations for these types of optimization problems, while for GA it is probably exponential, because it needs many more generations and iterations for larger sized problems.

Finally, RHC clearly performs the fastest, and blazingly fast compared to the other algorithms. The very simple nature of this problem allows for RHC to be very efficient because it only needs to change an integer of an array only containing 1's and 0's.

2.3 Flip Flops Problem

This problem attempts to maximize the number of non-consecutive integers [0-1], or "flip flops", in an array of size N.

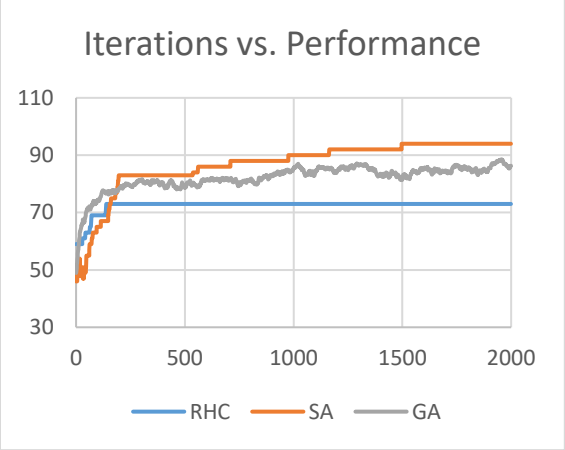

Figure 2.3a – Number of Iterations vs Maximum Count of Non-consecutive Integers (N = 100)

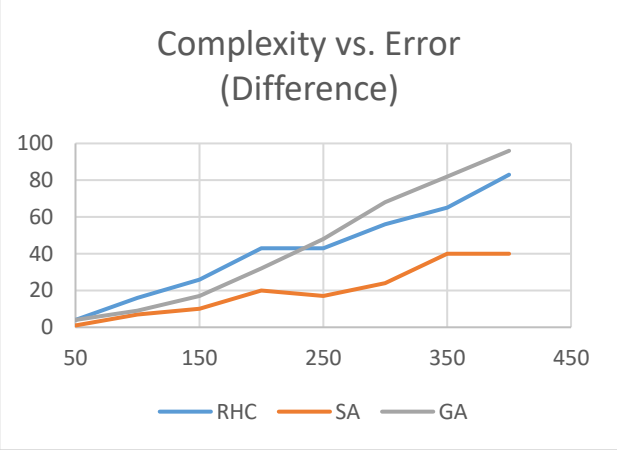

Figure 2.3b – Complexity (Size of Array, N) vs Error (N-max value obtained) (Iterations=2000)

Figure 2.3a outlines iterations vs the max amount of "flip flops", while Figure 2.3b focuses on the complexity (array size N = {50,100,……..400}) vs the difference in the absolute maximum value, N, versus the maximum value obtained by the algorithm (in other words, the error). SA it the best performer of the three algorithms and it appears that the margin between it and the other two algorithms would increase over more iterations. Looking at Figure 2.3a, RHC appears to completely plateau after the 100th iteration, while GA and SA trend upward throughout the iterations. This is because the function likely has a large basin of attraction which encompasses a wide part of the data space. RHC almost immediately get stuck in this local optima on every random restart which further evidences the potential of a large basin.

SA, on the other hand, starts out as the worst performer. However, as our neural networks analysis illustrated earlier, the SA is given a chance to explore the data because of its temperature component. It gets stuck in local minima like RHC but is able to escape. This happens frequently on the SA curve, which explain the small "bumps" in improvement.

Additionally, SA does better than GA because there is no apparent underlying structure and the data is not diverse. It also has the fastest convergence time. The way Abigail structures RHC likely causes it to degrade in efficiency over more iterations, as a random integer must be chose for each iteration.

Table 2.3c – Count Ones Convergence Time

At 1500 Iterations	
RHC	41 ms
SA	14 ms
GA	929 ms

3 – Conclusion

RHC, SA, and GA are all interesting randomized optimization algorithms. They can be utilized for very complex and structured functions (GA), as well as be utilized for their efficiency in optimization problems (RHC and GA). In my opinion it is likely the best to utilize these type of algorithms where a) there is no concrete solution to a particular function and randomization would be an advantage, and b) in combination with other learning techniques (like NN's in part 1).