An Analysis of Randomized Optimization

1. The Algorithms

In this analysis we will look at four randomized optimization algorithms: randomized hill climbing (RHC), simulated annealing (SA), genetic algorithm (GA), and MIMIC.

RHC chooses a random starting solution. Then, a neighboring solution in the hypothesis space is sampled. If it is better, it is saved as the new best solution. This proceeds until a local optimum is hit. Then, the process is restarted at a new random solution. If this iteration achieves a better solution, it is saved. The random restarts are one solution to the nemesis of hill climbing: local suboptima.

SA operates similarly, with some key differences. SA also takes in a starting temperature and a cooling rate. SA samples the neighbors of the randomly selected starting solution and saves a better solution. If the solution is worse, the temperature dictates the probability the new solution is saved anyway. As successive better solutions are found, the temperature is decreased by the cooling rate, so the algorithm tends to stay in ranges where it keeps finding better solutions. This is another solution to local suboptima.

GA functions quite differently from RHC and SA. GA takes in a sample size, a number of "mates", and a number of mutations to make per iteration. It generates a sample of solutions from a uniform distribution of the hypothesis space. A fitness function is applied to the sample and the N best solutions are chosen as mates. M mutations are made in each pair of these mates to accomplish better solutions. Obviously, $N < S$ so that new individuals can be added to the "gene pool". This requires fewer iterations, but more complex ones.

MIMIC works by taking a sample of solutions in the hypothesis space. It applies a fitness function much like GA and takes the median fitness of the sample. A new sample is now taken from the population with fitness greater than the previous iteration's median, and the new median fitness is saved. This requires much more computation per iteration, but guarantees that the algorithm will not get stuck at local suboptima, something that none of the other algorithms can do.

2. Randomized Optimization of Neural Networks

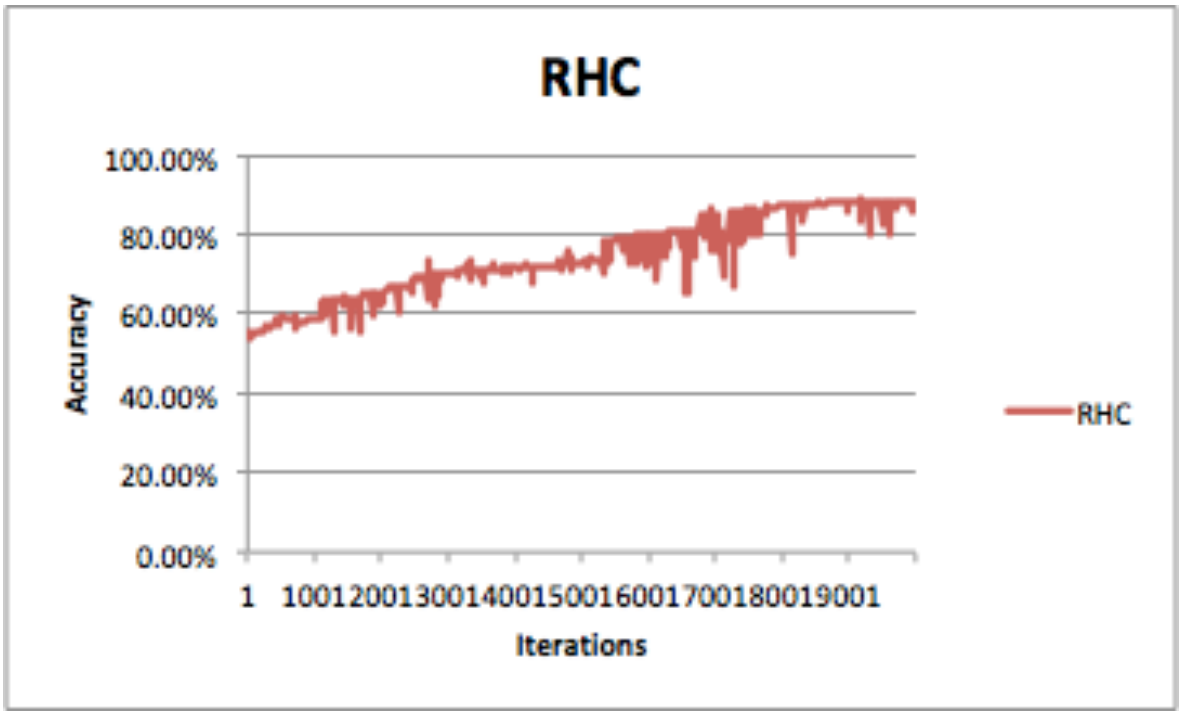
RHC, SA, and GA can all be applied to train the weights in neural networks. For this analysis, we will compare these algorithms' performance to the supervised neural network algorithm used in project 1.

We will test these algorithms on the spambase database. This database offers 4601 instances, each with 57 attributes. Each attribute corresponds to a word that potentially implies whether or not the email is spam and its count. This problem is much more exciting than the one offered by either the iris or wine datasets due to some interesting restrictions. A spam filter must not commit any false positives. This would result in a possibly important email being sent to spam. We will look at the global accuracy of these algorithms. In this context, overfitting will be defined as when the algorithm commits at least one false positive.

For our baseline we will use the classification of every instance as the most common class in the population. In this problem, that is the email is not spam. This heuristic shows us that 60.6% of the emails are not spam.

Our supervised neural net from project 1 will have 30 hidden layers, L = 0.3, M = 0.2, and 2500 epochs. Running this algorithm accomplishes 92.4% accuracy after a training time of 1827.8 seconds. Below, we will look at how randomized algorithms stack up against these baselines.

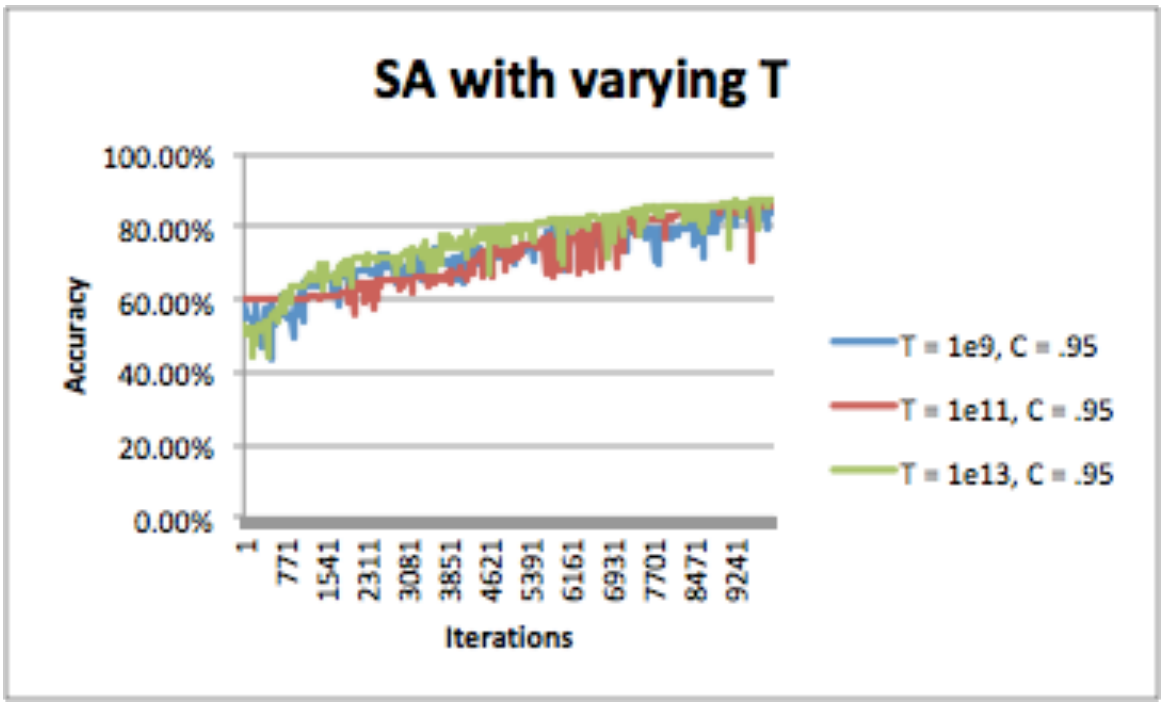
2.1 Randomized Hill Climbing

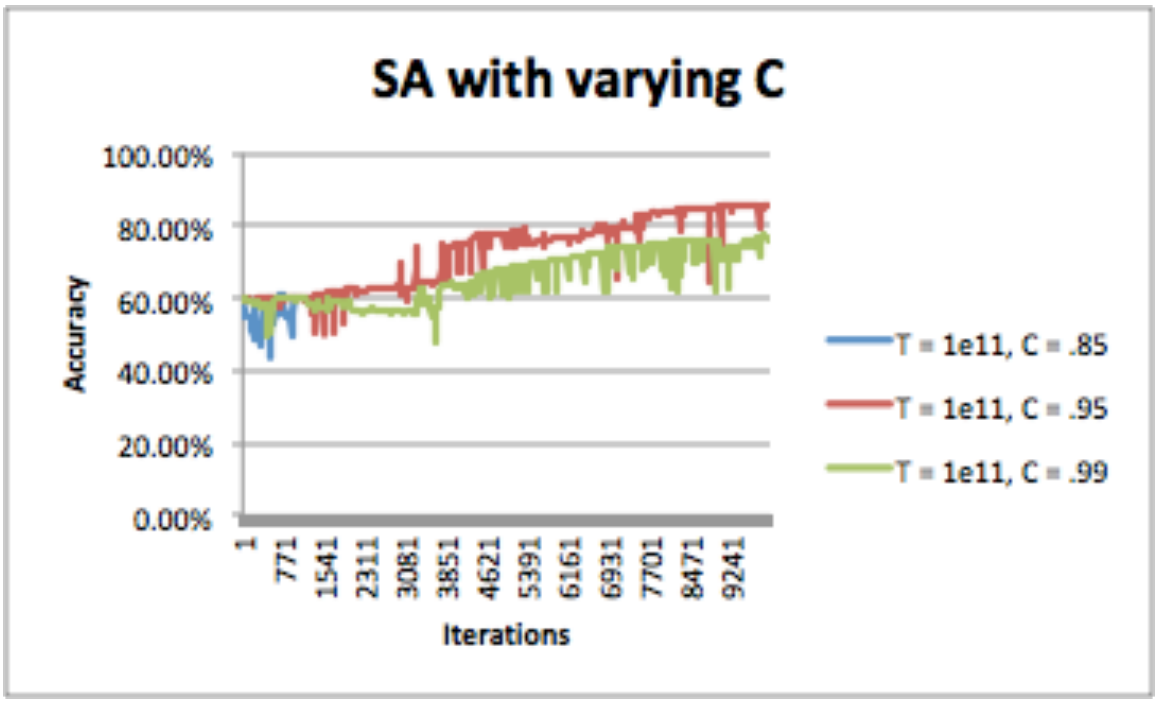


RHC requires the max number of iterations before stopping, as otherwise it will just continue to restart and try over and over again. For this analysis we chose 10000 iterations. RHC achieves 88.1% accuracy after 965.8 seconds. At this rate, this algorithm performs worse than the supervised neural net. . The graph gives some idea as to the characteristics to the spambase database. The diminishing returns of iterations, as well as the almost monotonic increase in accuracy over the range of iterations, seems to suggest that either there are many local maxima approaching a largest maxima, or there is one single global maxima and that the search space is continuous. The spambase database, with 57 attributes per instance, has an incredibly large search space, so the possibility of a global maximum is interesting. The random spikes in RHC must be due to noise in the data, but the noise is approximately gaussian. Therefore, minimizing the squared error will increase accuracy in general, but not necessarily every iteration.

2.2 Simulated Annealing

SA takes in the max iterations, the starting temperature, T, and cooling factor, C. We tested over 10000 iterations with T in the set {1e9, 1e11, 1e13} and C in the set {0.85, 0.95, 0.99}. While changing C, T was held at 1e11, and C was held at .95 while changing T.

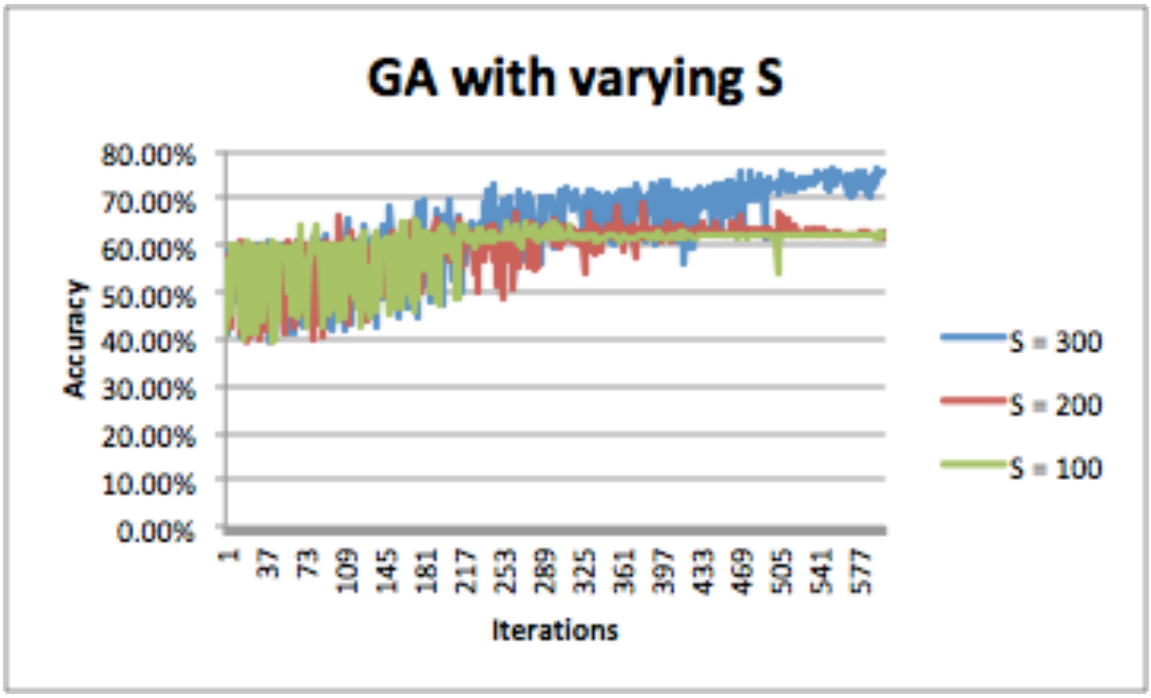




Of these combinations, a cooling factor of .95 is clearly dominant in both its quick acquisition of an accuracy of 80% and achieving a final accuracy of 84.1% accuracy. As for starting temperature, T = 1e13, does the best. Each combination completes training in between 900 and 1000 seconds. RHC and SA are very similar, especially since the spambase database does not seem to exhibit the characteristics that would allow SA to improve visibly upon RHC. As seen, the tendency for the solution to decrease in accuracy actually slows the convergence to a single value since it seems that the spambase database is smooth to a global maximum or has many maxima with approaching some global max. C represents SA's difference from RHC, and one would expect to see lower values of C performing more like RHC, so as C decreases, we should see a decrease in convergence time. However, the random nature of the algorithms does not allow this relationship to be seen after only one trial. Many trials would need to be done and averaged to show this relationship.

2.3 Genetic Algorithm

GA takes in the max iterations, the sample size, S, the number of mates, N, and the number of mutations per iteration, M. We used S (generation size) = {100, 200, 300}, M (number to mate in each generation) = 100, and N (number of mutations per generation) = 10.



Varying S shows some interesting characteristics of the data. Changing S changed where the algorithm converged. When S = 200, 100, the algorithm converges to the same value, but this is probably a coincidence. A larger generation size represents the ability to escape local suboptima. This testing reveals that there are certainly more than one local maximum. However, the greedy RHC and similar SA algorithms

show that there is only a small number of optimum and that the space is smooth. Varying M is similar to the momentum in our supervised learning algorithm, increasing it increases the time to converge, but increases the chance of finding better optima. Changing N is similar to the learning rate. However, the nature of GA allows it to handle non-differentiable functions. Each run took over 2000 seconds however to accomplish a much worse accuracy than our supervised neural network.

3. Optimization Problems

In this section, we will compare the previously analyzed algorithms as well as MIMIC. To see each algorithm's advantages and disadvantages, we looked at three different problems: Four Peaks, Flip Flops, and Travelling Salesman.

MIMIC

3.1 Four peaks

Four peaks is a function that that takes a bitstring of length N and a trigger position T. It maximizes at those bitstrings that have a series 0s or 1s up to the trigger point, and then the complementary bit contiguously until the end of the string, where the absolute maximum is 2N – T - 1. Two other local maxima can be found when the bitstring is all 0s or all 1s. We ran this algorithm with N = 85 and T = 8.

Algorithm	Parameters	% Suboptimal (out of 100, >=85)	% Optimal (out of 100, =161)	Time
RHC	1000 iterations	100	0	.14
SA	1000 iterations T = 1e13 C = .95	40	60	.18
GA	500 iterations S = 300 M = 100 N = 20	100	0	.11
MIMIC	500 iterations S = 200 K = 5	45	55	4.2

All algorithms accomplished at least a suboptimal maximum. SA and MIMIC performed the best. More trials would need to be done to see if this is coincidence. Only GA found solutions that were not 85 or 161, due to the nature of the algorithm. We would expect MIMIC with more iterations to have a better performance than the others, as its algorithm guarantees achieving the global optima in finite iterations.

3.2 Travelling Salesman

The travelling salesman problem used here consists of a graph with 50 vertices with randomized edges in the set of the complete graph with randomized weights.

Algorithm	Parameters	Average (out of 100)	Time
RHC	1000 iterations	.11	.20
SA	1000 iterations T = 1e13 C = .95	.11	.16
GA	500 iterations S = 300 M = 100 N = 20	.15	.58
MIMIC	500 iterations S = 200 K = 5	.08	33

GA and MIMIC show their increased computations per calculation, but GA accomplishes the best score. With increased iterations the differences in each algorithm would be more pronounced. The search space is simply too large for the narrow-minded RHC and SA to accomplish good scores. With more iterations, MIMIC is expected to have performed better.

3.3 Flip Flop

Flip Flop is a function that returns the number of times bits alternate in a bitstring, including the first bit (nothing to anything is a flip). “0011” would return 2. “0101” would return 4. The optimal solution produces a constantly alternating bitstring with a return value equal to the length of the string. The function has a large number of local maximums, with each bit as a separate attribute in a set of instances.

Algorithm	Parameters	Average (out of 100)	Time
RHC	1000 iterations	146.5	.23
SA	1000 iterations T = 1e13 C = .95	177.55	.15
GA	500 iterations S = 300 M = 100 N = 20	146.5	.23
MIMIC	500 iterations S = 200 K = 5	153.95	25.87

Only SA seems to perform well on this problem. This is due to the nature of the problem. The search is space is small where neighbors are very related. But there are many local maxima. Therefore SA's advantage over RHC is able to shine. MIMIC shows its computational complexity again, and the number of iterations required to have it score as well as SA would be many more and require much more time.