An Analysis of Random Local Search Algorithms

Abhinav Tirath

In this study, we compare and contrast three local search algorithms: randomized hill climbing, simulated annealing, and genetic algorithms. We first analyze the performance of each algorithm on two optimization problems. We conclude our analysis by using each of these algorithms in place of backpropagation in a neural network to find ideal weights and analyzing the results.

I. Problem Description

A. *Traveling Salesman Problem*

The first optimization problem is simply finding the optimal route given a list of locations and distances between these locations. This is represented by a salesman, who has several locations in which he or she must make transactions. Hence, it is called the Traveling Salesman Problem. An optimal route is one in which each location is visited with the shortest total distance traveled. By the nature of the problem, a small change in the order of locations visited can drastically affect the total distance traveled. As a result, this problem's evaluation function will have many local optima.

Figure 1. Displays the optimal route of visiting the 15 largest cities in Germany. Taken from *Wikipedia*.

This problem is interesting because it represents something that nearly everyone encounters in everyday life: finding an optimal route to multiple locations. With small adjustments, we could use this problem and its solution to help people optimize their time. In addition, companies could use the results to make their employees more efficient and, therefore, make more money.

B. *Count Ones Problem*

The second optimization problem is even simpler. It takes a bit string of a given length and determines the number of ones in it. As such, the only local optimum for the evaluation function will also be the global optimum, which is where the bit string consists of only ones.

This problem was primarily chosen due to the sharp contrast in the number of local optima. As a result, it is interesting because it represents any problem that has only one local optimum. Many problems have one local optimum, so the results from this problem can be applied to a wide range of problems.

II. Traveling Salesman Analysis

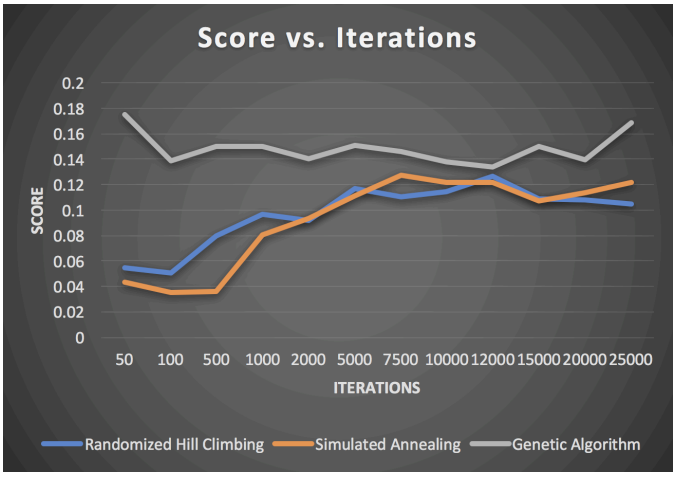

Figure 2. Shows each algorithm's score on the evaluation function vs. the number of iterations.

It is very clear that genetic algorithm performed the best on this problem. Even with just 50 iterations, the genetic algorithm had already converged and had a very high score.

Furthermore, simulated annealing and randomized hill climbing mirror one another quite well in this problem. They both start with very low scores and begin to increase as we increase the number of iterations. Both algorithms converge at about 5,000 iterations, but simulated annealing performs slightly better both overall and with the maximum amount of iterations (25,000). We can also see that the performance of randomized hill climbing gradually rose, whereas the performance simulated annealing had a sharp spike. This spike is likely due to

chance; simulated annealing simply chanced upon a good input.

Genetic algorithms performed much better on this problem than its counterparts because the problem has many local optima. Both simulated annealing and randomized hill climbing will get stuck in a local optimum. Randomized hill climbing will assess this value as ideal, and simulated annealing may struggle to get out of it, especially if the temperature is low. On the other hand, genetic algorithms will avoid getting stuck in these local optima because two different points can 'reproduce' to make a new point that may be on track for a different optimum. As a result, genetic algorithms will explore different hills and peaks more than the other algorithms.

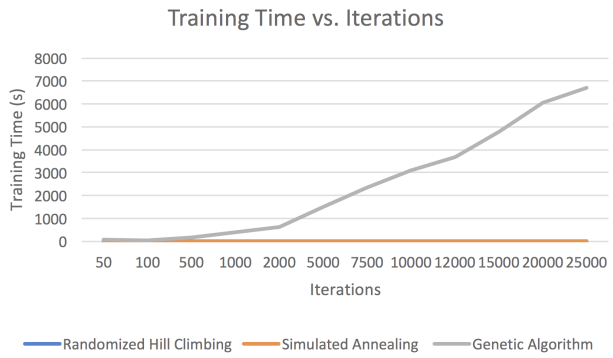

Figure 3. Shows each algorithm's training time vs. the number of iterations.

Figure 3 indicates, however, that there is a downside to genetic algorithms. As we increase the amount of iterations, training time for simulated annealing and randomized hill climbing stay relatively low, taking less than 30 seconds with 25,000 iterations. However, the genetic algorithm's training time is drastically more than those of the other algorithms, taking over 11 minutes to train with 25,000 iterations. However, we must remember that the genetic algorithm converged at a very low amount of iterations, and the training process was about a minute for 100 iterations.

The only potential change we could make to increase the score of the genetic algorithm would be to increase the population of points we are tracking at each stage. Still, since the algorithm converged so quickly, this change may have no effect at all on the algorithm's performance. This quick convergence also indicates that we can decrease the amount of iterations to decrease training time with little effect on performance.

To improve the performance of simulated annealing and randomized hill climbing, we would increase the amount of iterations. Although it appears as if they both might have converged, we might be able to improve performance with a few more iterations. Another way we could improve the performance of simulated annealing would be to decrease the rate of temperature decrease. Though this will increase their training times, their training times are already so low that it would not take much longer.

III. Count Ones Analysis

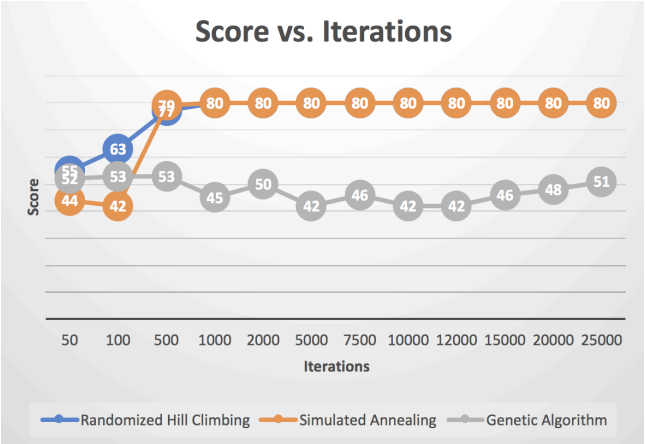

Figure 4. Shows each algorithm's score on the evaluation function vs. the number of iterations.

In this case, simulated annealing and random hill climbing clearly outperform the genetic algorithm. In fact, both of the former algorithms converge to the global optimum value at 1000 iterations. Similar to the last problem, randomized hill climbing gradually increases, whereas simulated annealing simply chances upon a good input and has a sharp spike.

The genetic algorithm, in this case, performs much worse. Similar to the previous problem, increasing the amount of iterations does not seem to make much of a difference; the score does not vary much after 50 iterations.

Simulated annealing and randomized hill climbing perform very well on this algorithm because the only local optimum is the global optimum. Both algorithms will essentially focus on climbing the hill until they converge to the global optimum. The genetic algorithm, however, does not do this. In this case, the 'reproduction' part of the algorithm hinders rather than helps it. The algorithm does not give all of its effort to 'mutating,' or climbing the hill.

The genetic algorithm, however, puts more focus on 'reproduction,' rather than just 'mutating' and climbing the hill as the other algorithms tend to do. This results in its performance being significantly lower.

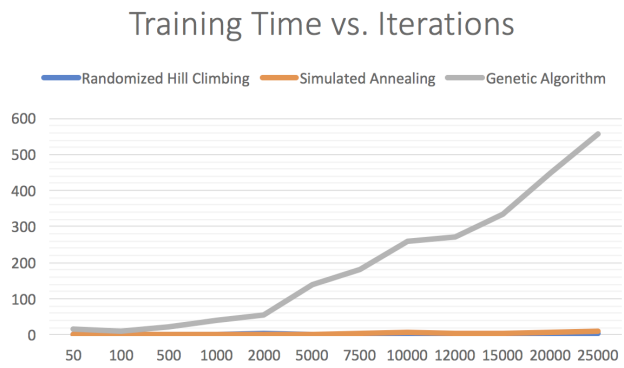

Figure 5. Shows each algorithm's training time vs. the number of iterations.

This graph is very similar to Figure 3 for the Traveling Salesman problem. We must note that all the algorithms had converged by 1,000 iterations, so the training time with 25,000 iterations is not the most relevant. With 1,000 iterations, the genetic algorithm takes 41 seconds to train, simulated annealing takes two seconds to train, and randomized hill climbing takes only one second to train. We note the trend that the genetic algorithm training time is significantly greater than those of the other algorithms.

In order to improve our test, we would reduce the maximum number of iterations to 2,000, as all three algorithms seem to converge quite quickly. We would not modify simulated annealing or randomized hill climbing, since they found the ideal solution in just 1,000 iterations. To improve the performance of the genetic algorithm, we would increase the population size, and increase the proportion of 'mutations' to 'reproductions.'

IV. Neural Network Analysis

As noted beforehand, we will now use each of our three algorithms in place of backpropagation to train a neural network.

A. Dataset Description

Our dataset used describes 10,000 separate credit card transactions, classified into two categories: fraudulent (represented by 1) and not fraudulent (represented by 0). The data is highly unbalanced; only 493 of the transactions are classified as fraudulent, as indicated in Figure 6.

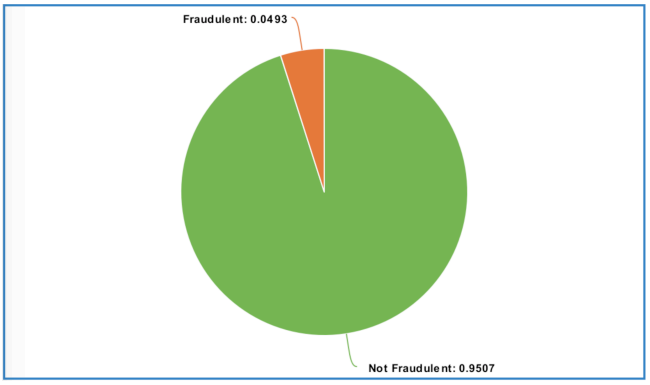

Figure 6. Displays the imbalance of the credit card data.

The data were collected and analyzed by a research collaboration of Worldline and the Machine Learning Group of ULB. This dataset has 28 features that describe each transaction, and no cell is blank. The number of instances in the dataset was reduced from about 243,000 to 10,000 in order to make the data more balanced and reduce time of computation.

The features are all numerical inputs labelled V1, V2, ... V28, and they are the result of a PCA transformation. Unfortunately, the original features and a further description of the data cannot be provided, due to confidentiality issues. In order to use this dataset with our optimization algorithms, we need a binary input, so we modified the input data. We determined the median of each feature, and replaced each input with a 0 if it was below or equal to the median or a 1 if it was greater than the median. Finally, we divided the data into 80% training data and 20% test data.

This problem is interesting because credit and debit card transactions account for over 50% of all transactions in today's world. For this reason, it is essential that payments be assessed as fraudulent or not fraudulent as soon as possible. If we fail to do so, customers will be charged for purchases that they did not make, and credit and debit card companies will fall to the wayside with lawsuits and chaos. A solution to this problem will ensure the safety of consumer money and credit card use.

B. Graphs and Analysis

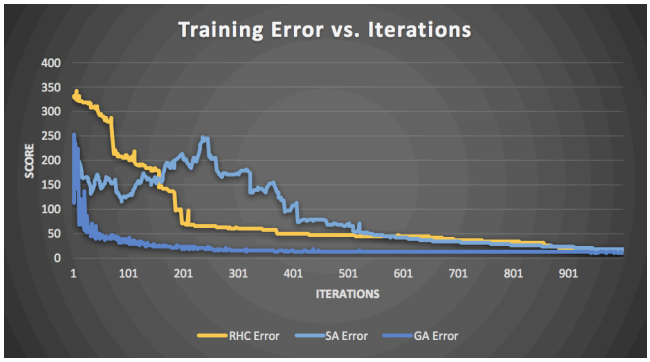

Figure 7. Displays the total error of the training set versus the number of iterations.

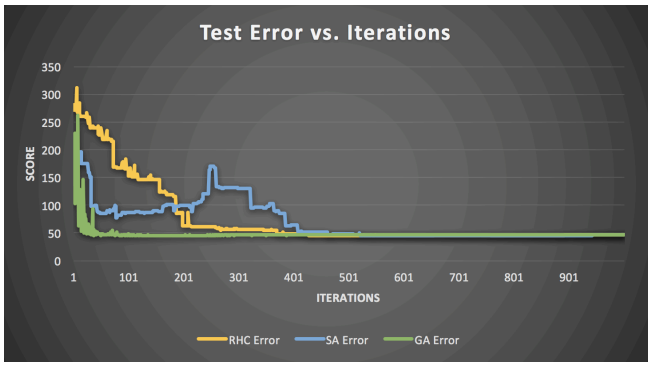

Figure 8. Displays the total error of the test set versus the number of iterations.

We must first note some peculiarities with our results. Although the test error mostly follows the training error well, it is clear that the training error for simulated annealing increases drastically, while the test error does not increase nearly as drastically. Additionally, randomized hill climbing has lower test error than training error in certain parts. Both these results can be partially attributed to the possibility that the test data is by chance somewhat easier to classify than the training data. Still these results are peculiar and should be revisited in a separate investigation.

Moving on, we see once again that the genetic algorithm converges in many less iterations than do the other algorithms. The testing error converges after just 50 iterations, although the genetic algorithm's training error continues to decrease after 50 iterations. As a result, continually increasing the iterations after this results in high variance.

As we have noted before, we continue to see that randomized hill climbing tends to gradually decline over iterations. We also observe that simulated annealing is much more erratic and makes many 'jumps.' It is also worth noting that randomized hill climbing has by far the worst performance with a very small amount of iterations.

However, it does appear that all three algorithms eventually converge to the same test error. We will examine this in more depth now.

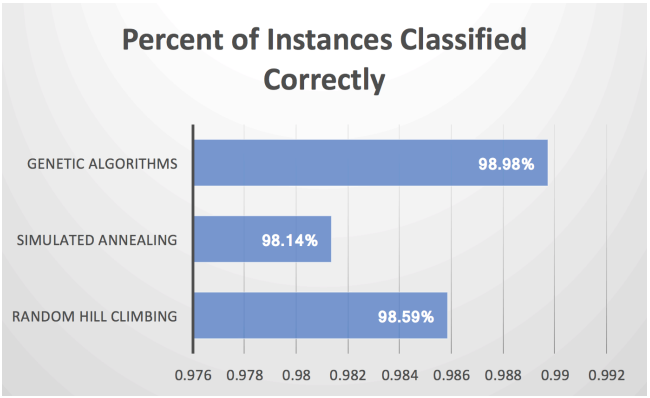

Figure 9. Displays the total test error of each algorithm with the maximum number of iterations.

As we can see in Figure 9, the overall scores from each algorithm were within a range of .84%, with genetic algorithms being the highest and simulated annealing being the lowest. From our analyses in the last sections, we predict that the optimization function in this case had several local optima.

In our previous research paper, we found that the neural network trained with backpropagation had 99.5% accuracy. This value is significantly higher than any of our results from the optimization algorithms. As we stated in the introduction, we modified the input data for our local optimization algorithms to be strictly binary rather than numerical. We posit that our local optimization algorithms did not perform as well as backpropagation because the input data did not give them as much information.

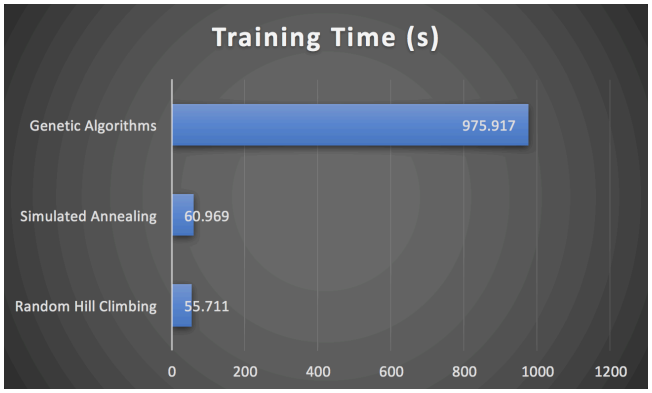

Figure 10. Shows each algorithm's total training time.

Similar to what we have seen before, the genetic algorithm takes significantly longer to train than the other algorithm. In this case, the genetic algorithm took over 16 times longer to train than simulated annealing and even more for randomized hill climbing.

Overall, the best algorithm for this dataset and neural network was either the genetic algorithm or randomized hill climbing because there are tradeoffs between the two. With the genetic algorithm, we obtain the best results but a very long training time and higher computational expense. With randomized hill climbing, we obtain fairly good results with a very short training time. It is worth noting that the neural network trained with backpropagation was by far the most accurate.

Since all the algorithms converged at around 600 iterations in terms of test error, reducing the number of iterations would decrease the training time for each algorithm without severely affecting its accuracy. To improve the performance of randomized hill climbing, we could have a larger number of random restarts in each iteration. For simulated annealing, decreasing the rate of the temperature decrease would improve its performance. Lastly, we could improve the genetic algorithm's performance by increasing the populations size.

V. Conclusion

A. *Randomized Hill Climbing*

Randomized Hill Climbing performed surprisingly well on each of our problems; it is able to find a mediocre to good solution for all of them. It works especially well with problems with only one local optimum, the global optimum, and it also was generally the fastest algorithm to train. These are true because simply focuses on climbing the hill it begins on. An easy way to improve the performance of randomized hill climbing is to increase the number of random restarts.

B. *Simulated Annealing*

We expected simulated annealing to perform much better on the algorithms; it was nearly always outclassed, either in training time or overall score, by randomized hill climbing. Still, it performed very well on problems with only one local optimum, and it trained much faster than genetic algorithms. An easy way to improve the performance of simulated annealing is to decrease the rate of the temperature decrease. Although this increases training time, we believe that simulated annealing's performance would significantly improve if we were to change this factor.

C. *Simulated Annealing*

Simulated Annealing performed very well on our problems. Its greatest strength lies in finding very good solutions in problem spaces with many local optima, something with which the other algorithms struggled immensely. The training time, however, was always much higher than the other algorithms. We suspect this is because the algorithm is keeping track of and modifying several points at the same time, unlike the other algorithms. Still, this disparity was overrepresented since the genetic algorithm tended to converge much more quickly. An easy way to improve the performance of the genetic algorithm is to increase the population size, but this would further increase training time.

References

ABAGAIL. (2016). Retrieved November 4, 2018, from https://abagail.readthedocs.io/en/latest/

ABAGAIL. (2018, March 09). Retrieved from https://github.com/pushkar/ABAGAIL

Travelling salesman problem. (2018, October 16). Retrieved November 4, 2018, from https://simple.wikipedia.org/wiki/Travelling_salesman_problem