Predicting Diabetes using Neural Networks and Randomized Optimization

Kunal Sharma

GTID: ksharma74

CS 4641 Machine Learning

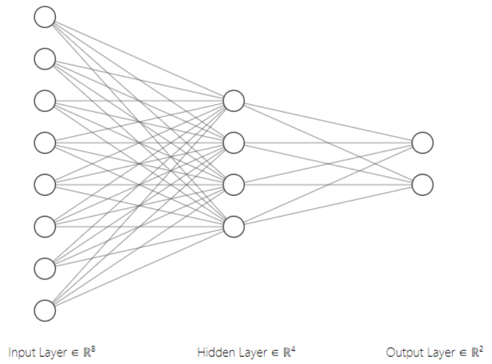
Abstract

This paper analysis the following randomized optimization techniques in detail: Randomized Hill Climbing, Simulated Annealing, and Genetic Algorithms. This paper is separated into two sections. In the first section, the aforementioned techniques will be used in place of backpropagation to find good weights for a neural network that detects diabetes based on various diagnostic measurements. In the second section, the performance of the three techniques will be analyzed on two optimization problems: Traveling Salesman and Continuous Peak Test.

The dataset will be normalized so that features with higher magnitude will not weigh considerably more and so that training time will be reduced. The dataset will be split randomly to create a training set with 75% of the data and a test set with 25%. Particular attention will be given to hyper parameter choices and accuracy curves over training iterations.

The dataset was obtained from Kaggle. Python, Java, Eclipse and Jupyter Notebook were used for development. Packages used for data manipulation and visualization include Numpy, Matplotlib, Pandas, and Scikit. The ABAGAIL library was used to implement the optimization algorithms.

The neural network architecture that will be used includes 8 input neurons since there 8 features in the data, 2 output neurons each representing a binary classification, and 4 neurons in the hidden layer.



Section 1: Finding Neural Network Weights using Randomized Optimization Algorithms

1.1 Diabetes Dataset Introduction

Originally, this dataset is from the National Institute of Diabetes and Digestive and Kidney Diseases. The aim of this dataset is to use a pre-selected set of diagnostic measurements to predict whether a patient has diabetes. The patients in this dataset are females of Pima Indian heritages that are at least 21 years of age.

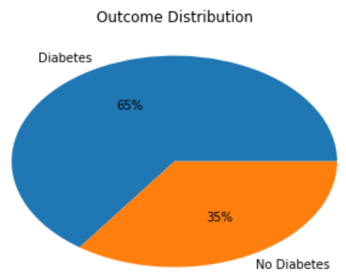
This dataset has a total of 768 rows, a single target column (outcome) and eight medical predictor column detailed in the chart below.

Feature Description

Pregnancies	Number of times pregnant
Glucose	Plasma glucose concentration a 2 hr in an oral glucose tolerance test
Blood Pressure	Diastolic blood pressure (mm Hg)
Skin Thickness	Triceps skin fold thickness (mm)
Insulin	2-Hour serum insulin (mu U/ml)
BMI	Body mass index (weight in kg/(height in m)^2)
Diabetes Pedigree Function	Diabetes pedigree function
Age	Age of Patient
Outcome	Class variable (0 or 1)

Outcome Distribution

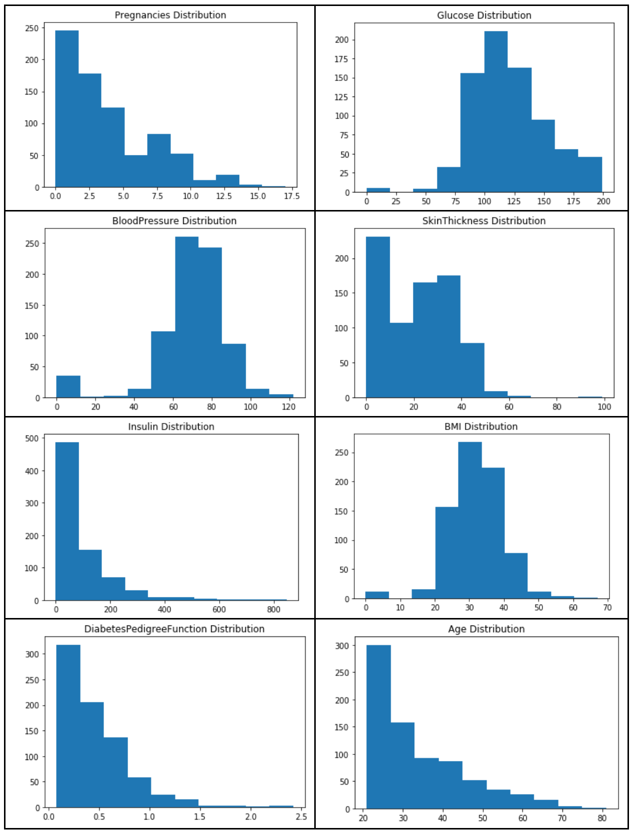
65% of the data represents patients diagnosed with diabetes and 35% represent patients without diabetes. This dataset is therefore unbalanced and we should expect our accuracies should be above 65% if we expect our models to be predictive.



Feature Data Distribution

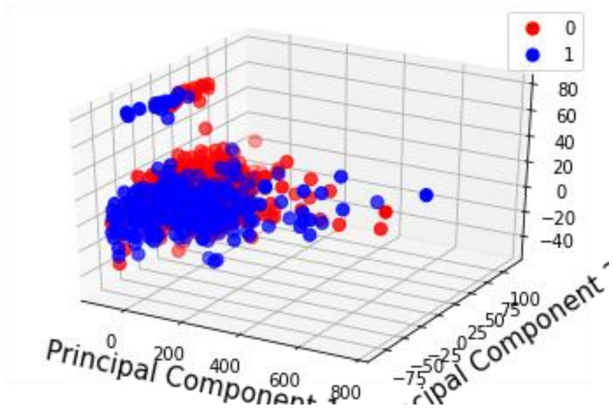
Glucose, Blood Pressure, and BMI follow a roughly normal distribution while the remaining features are typically skewed to the right.

Left to Right: Pregnancies, Glucose, Blood Pressure, Skin Thickness, Insulin, BMI, Diabetes Pedigree Function, Age



Principal Component Analysis

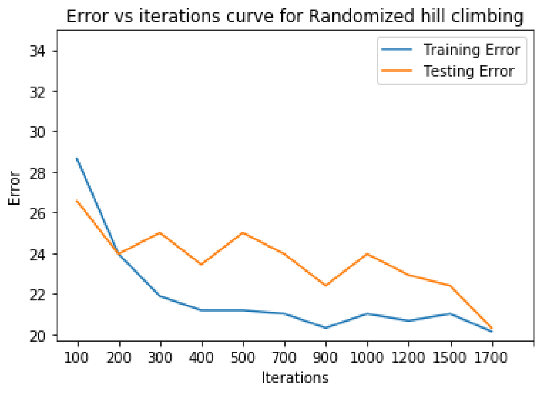
We can use Principal Component Analysis to reduce the dimensionality of the feature space from 8 to 3 dimensions so that we can visualize the data in the figure below. The clustering in the visualization is a good sign that our data can be separated and classified.



1.2 Randomized Hill Climbing

The randomized hill climbing algorithm builds on the simple hill climbing algorithm. In hill climbing, a point is chosen randomly and the fitness values of nearby points are compared. The point with the highest fitness is selected and then nearby points are compared again until the point reaches the top of the hill. The issue is that the algorithm often gets trapped at local peaks. Randomized hill climbing reduces the likelihood of getting stuck at a local optima by introducing multiple random restarts across the search space.

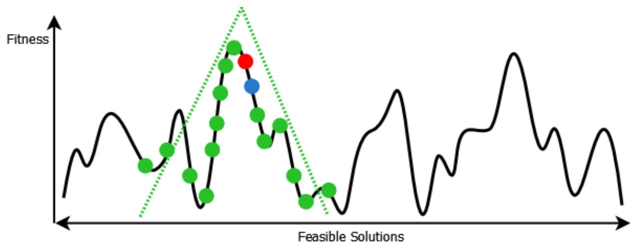
The next graph shows the error in training and testing over 1700 iterations. Both errors reduce over iterations. The training error reduces to 20.139% and the test error reduces to 20.312%. As expected the training error converges faster and has a (slightly) lower error than the test set.



Since the error between the test and training error is very close it is very likely that the random hill climbing algorithm was able to find the global maximum. The algorithm took 9.12 seconds to train and 0.001 seconds to test.

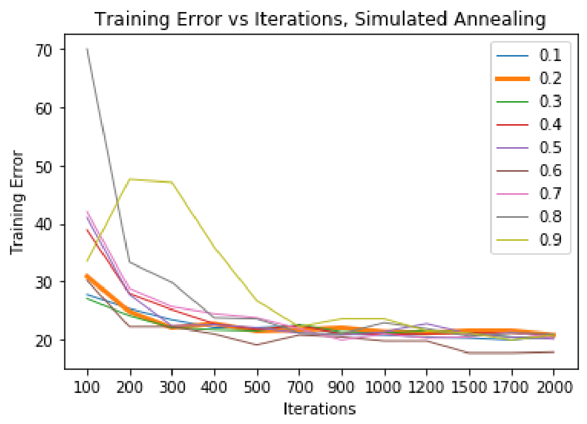
1.3 Simulated Annealing

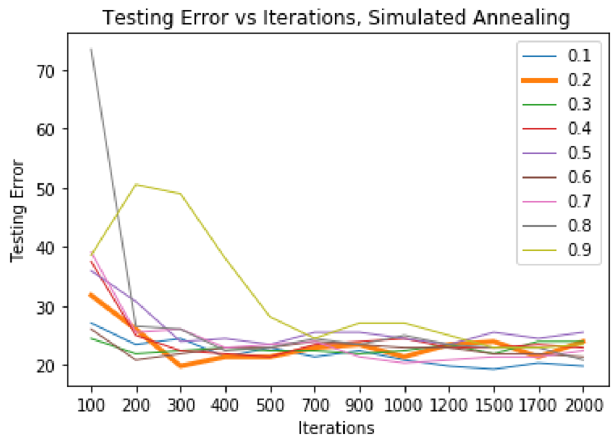
Another randomized optimization technique, simulated annealing, is based on the concept in metallurgy of annealing. In metallurgy, the molecules in a metal align after the metal is heated and cooled several times to increase the final quality and durability of the metal. Simulated annealing is also a hill climbing algorithm but goes further by applying this concept of annealing using probabilities and statistics. At every iteration, the algorithm decides if it moves up or down the hill based on probability values.



A temperature parameter determines the probability that the point will go up or down the hill (in other words “explore” or “exploit” the search space). The algorithm starts with a high temperature which decreases at some cooling rate. When the temperature is higher the algorithm is more likely to explore the search space and pick points farther and lower than where it is with the aim of finding a higher peak. When the temperature is lower, the algorithm is less adventurous and focuses on finding more points with a higher fitness score nearby.

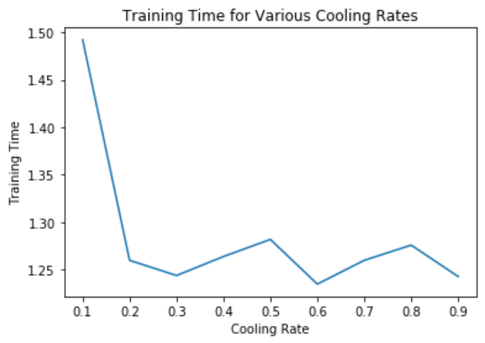
To find the best cooling rate we can run the simulated annealing algorithm for 2000 iterations with different cooling rates starting at 0.1 and incrementally increasing to 0.9 by 0.1. The training error and test error curves are illustrated in the following graphs.

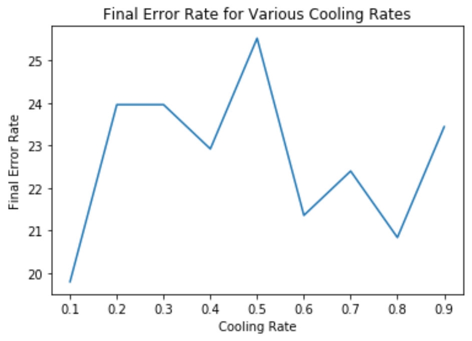




For all cooling rates, both the training and test error eventually converged to an error rate below 26%. For example, the cooling rate of 0.8 starts with a high test error near 70% on the 100th iteration but converges to an error of 20.83%.

There are two key criterion in selecting the optimal cooling rate: training time and accuracy which are graphed below.





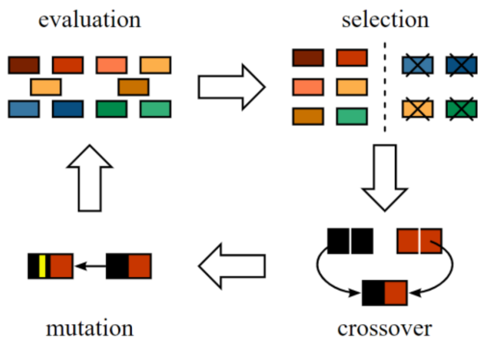
The cooling rate of 0.1 has the lowest error of 19.79%. However, this cooling rate also has a training time roughly 0.25 seconds greater than the others. It is important to note that after about 1000 iterations, the cooling rate of 0.1 consistently has the lowest accuracy. Thus, we will select 0.1 as the optimal cooling rate given its accuracy and consistency, despite a small trade off in training time (which is expected for a slower cooling rate).

We can infer that the algorithm’s search space doesn’t have a local optima since it has the lowest test error while its temperature is decreasing the fastest.

If we wanted to find an even better rate we can compare the performance of values near 0.1.

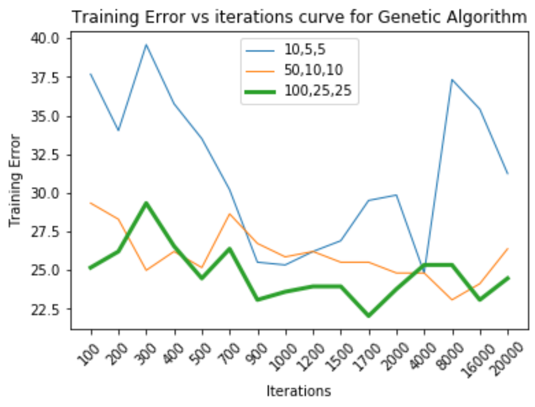
1.4 Genetic Algorithm

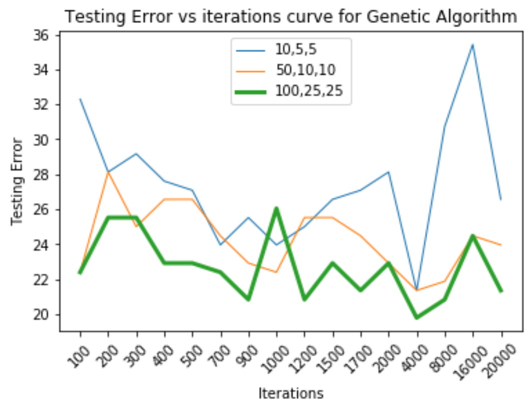
The genetic algorithm uses concepts of evolution including mating, mutations, and populations to take an evolutionary approach to randomized optimization. At the start of the algorithm there is an initial randomly generated population of individuals. Each individual in this population contains information that can be “mutated” or “mated” (crossed over with) other individuals. The algorithm is trained through iterations called generations. Mutations and mating (crossover) occur every generation and only the individuals with the highest fitness values are selected to breed the next generation.



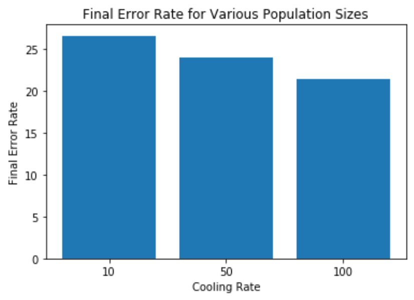
Source: http://www.jade-cheng.com

The following graphs plot the results of 20,000 iterations of the genetic algorithm with various hyper parameter choices including (in respective order) population size, mating count, and mutation count.



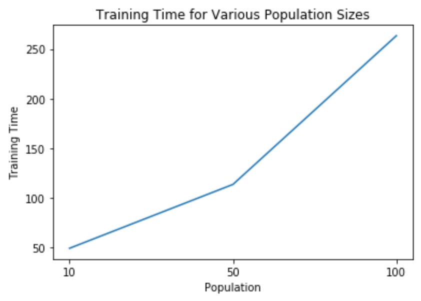


Despite undergoing 20,000 iterations (10 times the number of iterations than random hill climbing and simulated annealing), the genetic algorithm does not converge and even if given a significantly greater number of iterations it may not do so. The graph below illustrates the final error.



The algorithm with the highest population size hyper parameter outperforms the other algorithms with the lowest error of 21.35%. This conclusion is intuitive because a larger population means a greater likelihood for the algorithm to find the global maxima.

However, the tradeoff of a larger population is slower training time. The training time for a population of 100 is more than 5 times the training time for a population of 10.



With a problem like diabetes, it is logical to prioritize accuracy as over a few minutes of training. Thus, the genetic algorithm with the population of 100, mating count of 25, and mutation count of 25 is the best choice.

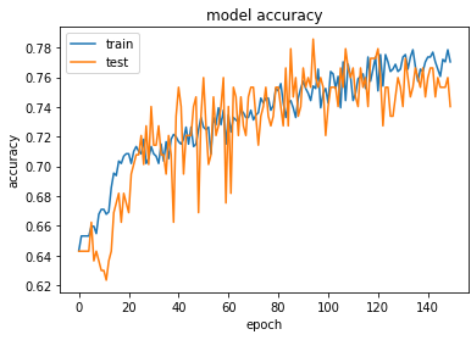
The results of this algorithm may change if the network is retrained given that the algorithm is randomized.

1.5 Conclusion for Randomized Optimization

All four algorithms for finding good weights for a neural network that predicts diabetes achieved accuracies with a range of only 1.56%

Algorithm	**Accuracy**	**Iter.**
Backpropagation	79.8%	100
Randomized Hill Climbing	80.21%	1700
Simulated Annealing	80.21%	2000
Genetic Algorithm	78.65%	20000

The following graph plots the training and test accuracies of the neural network that uses backpropagation to find weights (created for the last project).



Notably, with this algorithm, convergence occurs in about 100 iterations. While using backpropagation does risk getting trapped in local optima, it converges the fastest and most consistently. (This is likely why backpropagation is the most commonly used neural network training method in industry.) Thus, backpropagation can be considered the best algorithm for this dataset when accounting for both performance and training time.

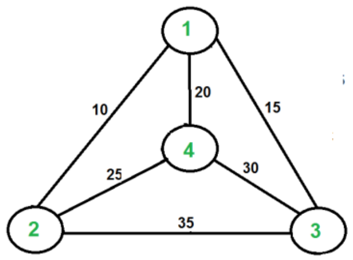
Section 2: Applying Randomized Search Techniques to Two Optimization Problems

The two optimization problems that will be covered in this section include the Traveling salesman and Continuous Peaks. We will solve these problems using the three randomization algorithms that we analyzed in section one.

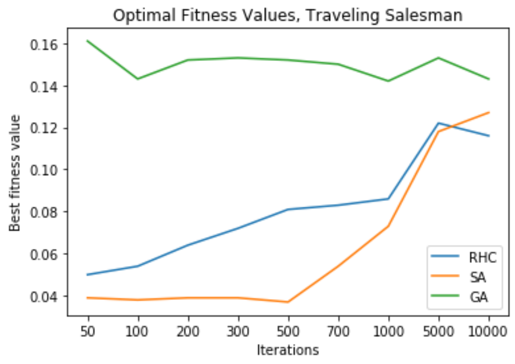
The best performing technique will be the one chosen based on fitness values as well as computation duration. To reduce the variance in our results, the best fitness value and computation time is averaged over five test runs.

2.1 Traveling Salesman Problem

In mathematics, the Traveling Salesman problem is classified as an NP hard problem. The Traveling Salesman problem provides a list of cities and the distances between each city and asks for the shortest route to visit all of the cities and return back to the first city visited. This problem is often represented with graphs where the vertices represent cities and the edges represent the distance between them.



We shall apply our randomized optimization techniques to see which performs the best on this type of problem using the Traveling Salesman problem built into the ABAGAIL library. The graph of the results is illustrated below.



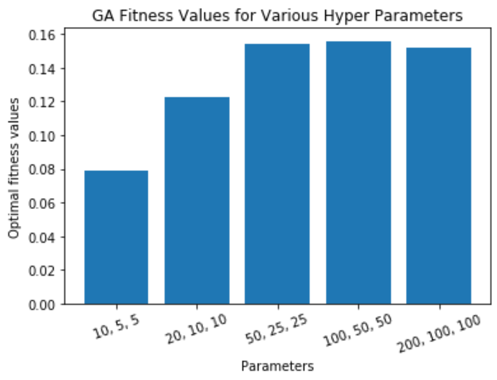
Fitness and test time on the 10,000th iteration:

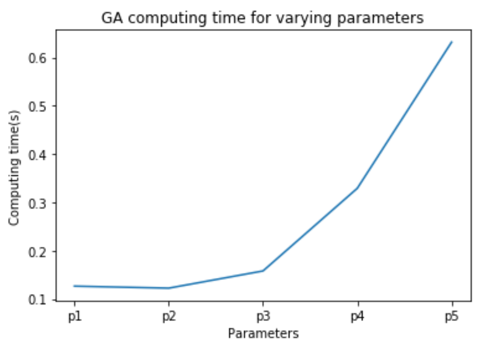
Algorithm	Fitness	Test Time(s)
RHC	0.116	0.005
SA	0.127	0.008
GA	**0.143**	3.796

It is evident that the Genetic Algorithm performs the best on the traveling salesman problem. It reaches the highest fitness value of 0.143 and it does so in the first 50 iterations while the other algorithms take over 5000 iterations to reach a fitness above 0.12. While it is slower, it converges to a higher fitness faster and thus should be selected as the best algorithm for this problem.

Genetic algorithms are not just a good approach for the Traveling Salesman problem but for most np hard problems because the search space is so complex and large that using procedural algorithms is not mathematically feasible but evolutionarily improving a randomized approach means less computationally costly solution. However, using genetic algorithms does not guarantee an optimal solution but it typically does provide one that is good enough.

To further improve the results of our genetic algorithm we can test different hyper parameter values. Below are the results of various hyper parameters: population size, mating size, and mutation size (respectively). 100, 50, and 50 are the parameters the result in the highest fitness score.

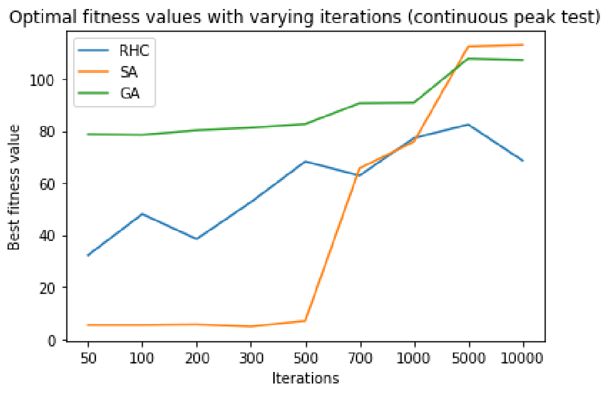




2.2 Continuous Peak Problem

One type of peak problem is the Continuous Peak problem. 0's and 1's can appear anywhere in a string. When there are greater than Y bits that are contiguous and all set to 1 or 0 then reward is given. The algorithm is searching amongst local optima for the global optima.

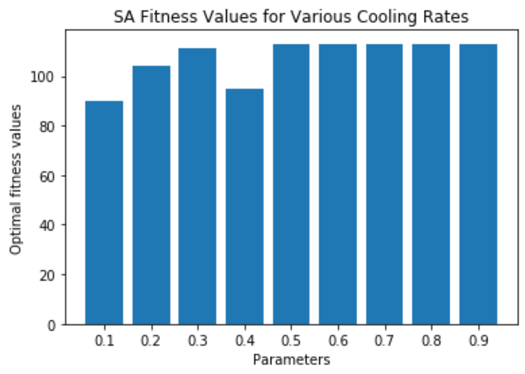
We shall apply our randomized optimization techniques to this problem and visualize our results with the graph below. The ABAGAIL library is set to use the parameters N = 50 and T = 5.

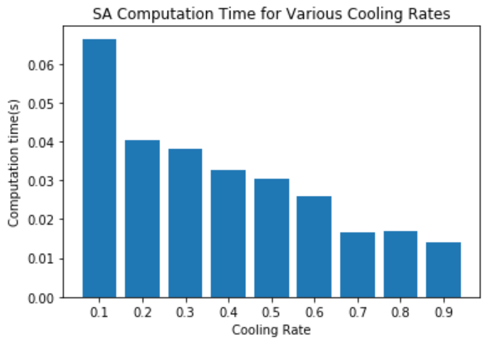


Algorithm	Fitness	Time (s)
RHC	68.6	0.003
SA	**113**	0.005
GA	107.2	0.914

It is evident that the simulated annealing algorithm performs the best on this type of problem with a fitness score of 113. It performs very well on time as well, with just 0.002 seconds longer than Random Hill Climbing it achieves a fitness that is almost double. The time that simulated annealing takes is less than a percent of the genetic algorithm, yet the SA fitness is still greater by 5.8 points.

We can further investigate this problem by varying the cooling rate of the simulating annealing problem as shown in the following graphs.





Performance is better when the cooling rate is higher. This makes sense because it means that the temperature will stay higher longer, and a higher temperature means that the algorithm is more likely to explore the search space. With this particular problem where exploration to find the global optima is the most important and directly results in reward, it makes sense why SA performs the best. In this case, a cooling rate of 0.5, 0.6, 0.7, 0.8 or 0.9 all result in the same fitness score. However, the cooling rate of 0.9 has the fastest computation time and is thus the optimal parameter for this problem.

3 References

Diabetes Dataset:
https://www.kaggle.com/uciml/pima-indians-diabetes-database

ABAGAIL Library:
https://github.com/pushkar/ABAGAIL