Random Search Report

An objective look at random search performance for 4 problem sets

Dudon Wai
Georgia Institute of Technology
CS 7641: Machine Learning
Atlanta, GA
dwai3@gatech.edu

Abstract: This report presents an analysis on the performance of 4 random optimization algorithms tested on three cost functions, of different types: "Continuous Peaks", "Knapsack" and "Travelling Salesman". The random optimization algorithms compared are: Random Hill Climbing, Simulated Annealing, Genetic Algorithm and MIMIC. Finally, the performance of random optimization in determining weights in artificial networks is compared with back propagation for the "Letter Recognition" dataset from Assignment 1.

Introduction

The purpose of this report is to explore a variety of random optimization algorithms in two parts: firstly by comparing behavior when applied to three cost functions, and secondly by comparing the performance of back propagation with using the optimization algorithms to find optimal weights for a feed forward neural network. In Part 1, the following problems were selected for their difference in nature, and wide spectrum of applications: Continuous Peaks, Knapsack and Travelling Salesman. In Part 2, the problem set is borrowed from Assignment 1 as it has already been evaluated with ANN and back propagation: Letter Recognition.

Optimization Algorithms

The following algorithms are compared and implemented using ABAGAIL, as provided in the class. The Java code was built using IntelliJ IDEA.

Randomized Hill Climbing (RHC)

Randomized Hill Climbing locates local optima by moving towards more optimal neighbors until it reaches a peak. With random restarts, RHC randomizes its starting position to locate other local optima, and selects the value with the highest value as the global optimum. RHC was performed using RandomizedHillClimbing in ABAGAIL on Java.

Simulated Annealing (SA)

The term Simulated Annealing originates from metallurgy where the ductility in metals are improved by heating to a higher temperature (below melting, above the annealing temperature where residual structural stresses are relieved) and then slowly cooled to maintain its structure. The algorithm, a function of initial temperature and cooling rate, strikes a balance between exploring new points and exploiting nearby neighbors in search of local optima. Initially, at high temperatures, the algorithm explores by randomly seeking new points and as it cools, it proceeds to evaluate neighbors for local peaks. SA was performed using SimulatedAnnealing in ABAGAIL on Java, using inputs temperature (1E11) and cooling rate (0.95).

Genetic Algorithm (GA)

Genetic Algorithm is inspired by biology in which the population evolves by iteratively mating and mutating parts to crossover the best traits and to eliminate irrelevant traits. A significant disadvantage of GA is that it does not handle a large hypothesis space, which is dictated exponentially by the number of attributes. SA was performed using StandardGeneticAlgorithm in ABAGAIL on Java, using inputs populationSize (200), toMate (100), toMutate (10).

Mutual-Information-Maximizing Input Clustering (MIMIC)

MIMIC algorithm, as opposed to most optimization algorithms, "remembers" previous iterations and uses probability densities to build structure of the solution space and find optima. MIMIC was performed using MIMIC in ABAGAIL on Java, using inputs samples (200) and toKeep (20).

Part 1: Apply Random Optimization to 3 Problems

Implementation

The existing cost function examples in the GitHub repository of ABAGAIL were used. Each algorithm was run using iterations of {100, 500, 1000, 2000, 3000, 4000, 5000, 10000, 50000, 100000, 200000} to observe how quickly the algorithms converge on the optima. The exception was that MIMIC was performed up to 5000 iterations, as the algorithm is highly computation intensive but primarily because the additional time it takes to build structure while learning enables the algorithm to learn with fewer iterations. For the purpose of comparison, the function value for MIMIC at 5000 iterations is extrapolated uniformly to highlight the performance of other algorithms at high iterations.

Each of these tests were run 10 times (that is, iterations =100 was run 10 times, iterations = 500 was run10 times, etc.) and the average function value is reported. The code was modified to accommodate the 10 repetitions, and in addition the code was modified to record the time to run each iteration. The data was then exported to Excel and analyzed. For further study, it is suggested to repeat for 100 cycles, and to report the convergence of each algorithm per problem set.

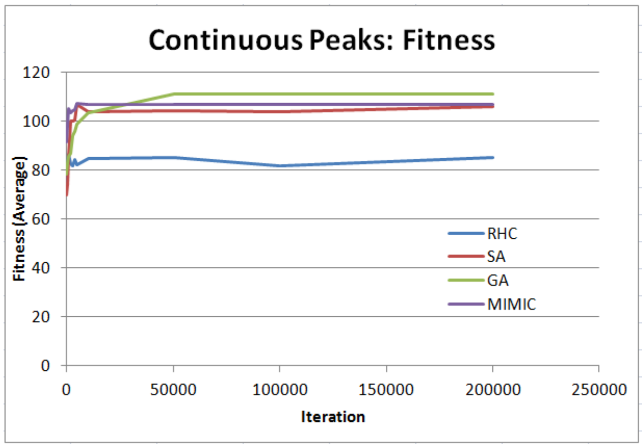
The default parameters in ABAGAIL were used for SA (temperature = 1E11, cooling rate = 0.95), GA (populationSize = 200, toMate = 20, toMutate = 10) and MIMIC (populationSize = 200, sampleSize = 20). It would be interesting to tweak the parameters of each algorithm to observe how the performance of each algorithm changes.

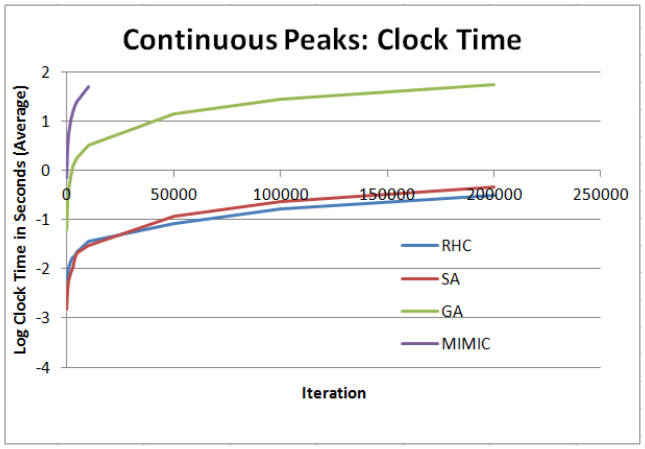
Optimization Problems

While already included in the ABAGAIL repository, the problem sets were specifically chosen from a list of common problems [1]. The problems were chosen for their uniqueness and broad applications.

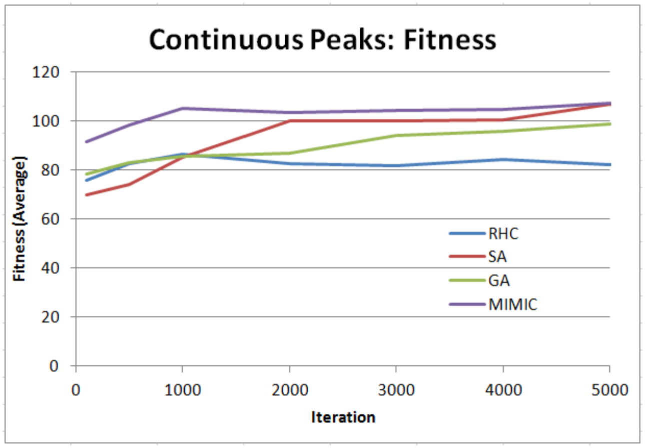
Continuous Peaks (SA)

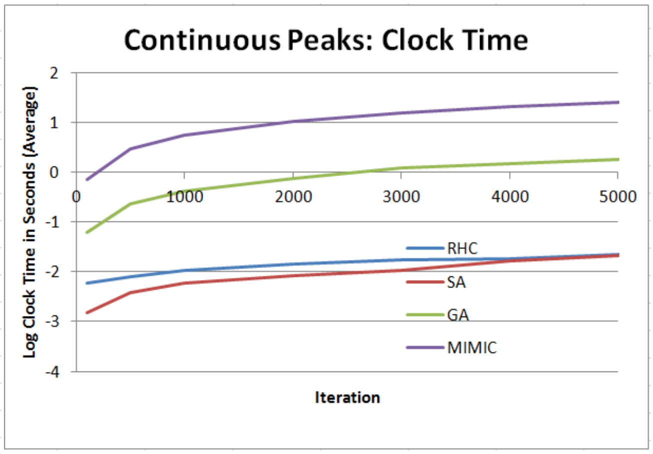
This problem set contains many local optima in a 1D space, similar to the example in lectures about determining the elevation of many peaks. Although this problem set was chosen for its simplicity, it highlights differences between random optimization algorithms well and applies to other examples like topography and optimization of surfaces. It is intuitive for a human to observe the global optima visually, however using random optimization and without processing each data point, the different performance between the random optimization algorithms perform is quite evident.





As the above charts show, genetic algorithm appears to perform best, at least for a large number of iterations. However, GA only begins to perform best around 30000 iterations, and overall SA/GA/MIMIC perform closely to 110. SA, in fact, performs the best at iterations below 5000 that are able to achieve 105 function value, as seen in the charts below. Additionally, SA achieves this comparable performance at least 2 orders of magnitude faster than GA and MIMIC.

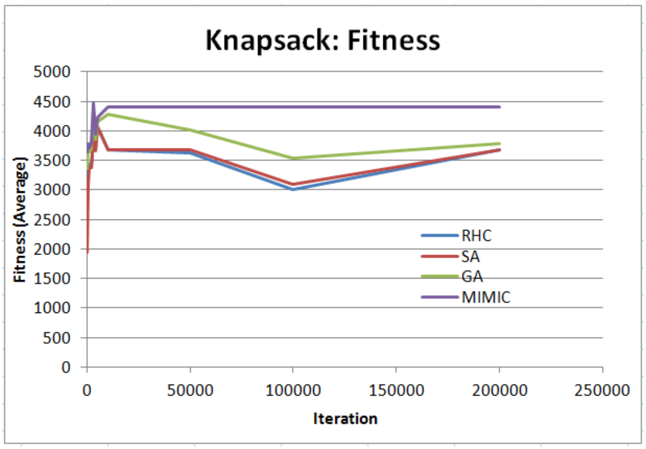


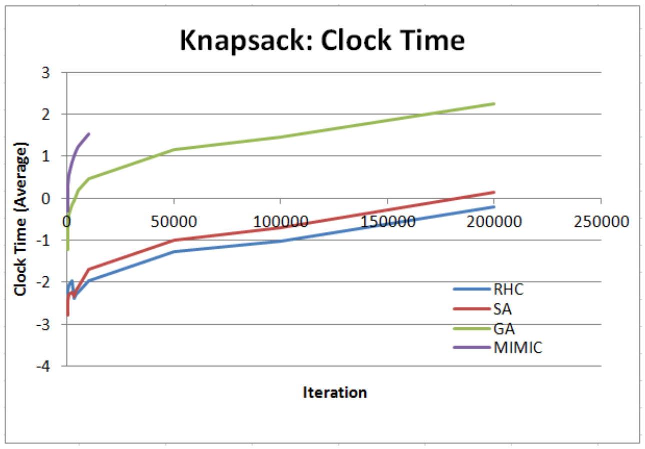


Simulated annealing performs well on the continuous peaks problem, as it is well suited for a large search space and for finding an approximate global optima, rather than a precise local optimum. Beginning in the exploring phase, SA randomly searches the space for optima and begins to cool and exploit the observed points by decreasing the probability of accepting worse solutions. Given that SA performs well with the default parameters, further study and additional time would allow a study of the impact of using temperatures of 1E9 to 1E12, and cooling temperatures in 0.05 increments between 0.05 and 0.95.

Knapsack Problem (MIMIC)

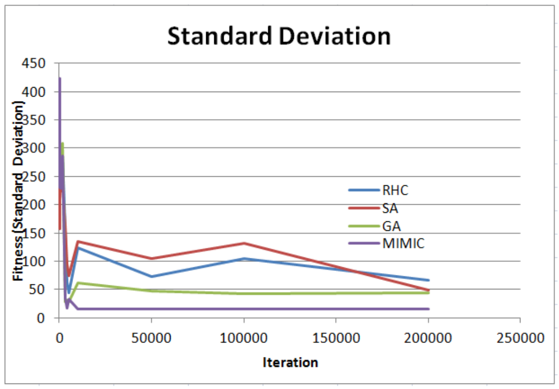
The Knapsack problem is a NP-hard (at least as hard as the hardest problems in non-deterministic polynomial acceptable problems) optimization problem with a set of constraints. The analogy lends itself from backpackers who try to maximize the occupying volume in a knapsack while remaining within the weight threshold such that the knapsack can be carried long distances. The problem is unique because there is no solution that is both maximal volume and lightest weight. This optimization problem can be modified to consider the number of items packed, but can also related to the “cutting stock” problem in which manufacturers have stock material and want to make the largest number of parts while reducing the amount of wasted/unused material. Further applications include maximizing profits for a business while considering environmental or social concerns.





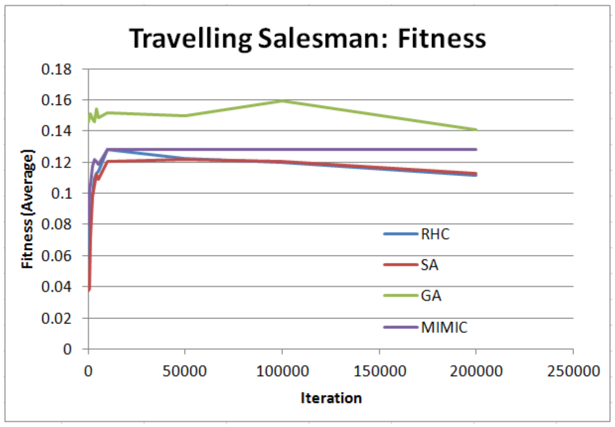
As the above charts show, the algorithms show peak performance around 5000 iterations, and begin to plateau at a lower function value for 50000 or more iterations. It is expected that the lower performance at 100000 iterations occurs due to a small number of repetitions (10) which is not enough to demonstrate convergence of the fitness function. This is demonstrated in the chart below where the standard deviation for the 10 repetitions increases at iterations = 100000.

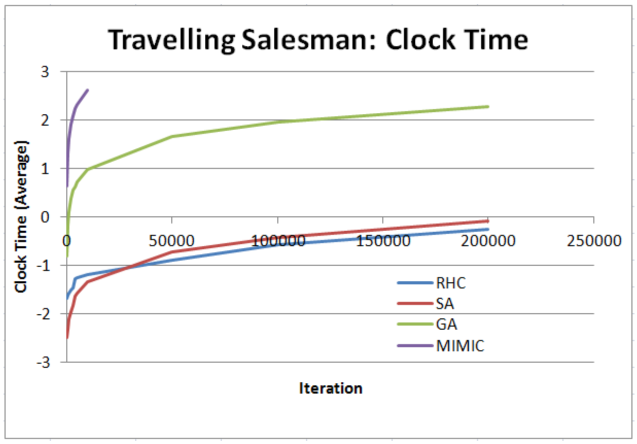
This problem set was selected to highlight the benefits of MIMIC. MIMIC consistently performs better than all the algorithms by roughly 10% and for iterations up to 5000 it has the lowest standard deviation. The low standard deviation (see chart below) indicates that between the 10 repetitions, MIMIC arrives at similar optima quite consistently, despite a different and random starting point. It is important to note though, that similar to the continuous peaks problem, the logarithmic clock time is considerably higher for MIMIC (again, as a trade-off for developing structure). The results are interesting, as MIMIC and GA typically perform better with continuous attributes because each successive generation interpolates or extrapolates from the previous. For this problem set, MIMIC may perform well for this type of problem as it can learn from previous searches and interpolate for the best option. MIMIC performance may be further improved by evaluating the performance when changing the population and sample sizes.



Travelling Salesman (GA)

The travelling salesman is similar to the knapsack problem, in which the travel distance between many locations is minimized (assuming each location is visited once). This has useful applications to the travel, transportation, shipping and logistics in which destinations rarely lie along a line and some extent of detours is required to visit all locations. In the knapsack problem, MIMIC and GA were not expected to perform the best because they typically do better for continuous functions, however MIMIC performed well. In this problem set, GA performs well and it is expected that MIMIC does not perform as well here because, as an NP-hard problem, the hypothesis space is too complex.





When visualizing the problem, the difference from the knapsack problem begins to show. The travelling salesman problem is a set of points in a 2D plane in which the cost function relates to the total distance travelled when all points are visited. MIMIC uses joint probabilities to develop an understanding of dependencies within the data, however for this problem, various optimal solutions may be significantly different because the order of the points visited matters whereas in the knapsack problem it did not. The comparison between problems is essentially the difference between a problem with vectors and scalars. It would be interesting to see the effects of varying the population, mate and mutate sizes for this problem set.

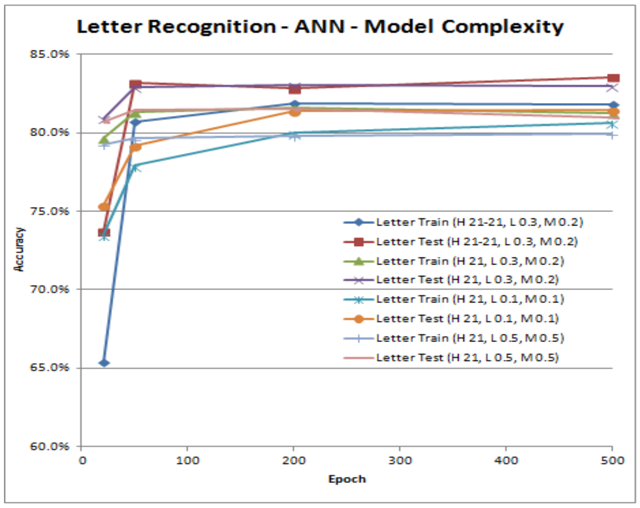
Part 2: Neural Network with Optimization Algorithms

Introduction

In Assignment 1, two datasets were evaluated when comparing supervised learning algorithms, including artificial neural networks. The purpose of Part 2 is to implement the random optimization algorithms with feed-forward neural networks, and compare the performance with back propagation from Assignment 1.

Dataset

The letter recognition dataset contains 26 classes (one for each letter in the alphabet), 16 attributes (position, length, statistical moments), and 20,000 instances of user-generated letters based on a variety of fonts. The size of the dataset allows for proper segmentation into training, validation and test sets, although not required in this assignment. Based on Assignment 1, the best neural network had an input layer equal to the number of attributes (16), an output layer equal the number of output classes (26) and a hidden layer with the number of nodes equal to the average of the input and output (21). The weights within the network were determined using back propagation, with the ideal parameters learning rate = 0.3 and momentum = 0.2. The peak performance of ANN using WEKA was about 85% accuracy.



Implementation

The dataset was modified from .arff file that listed the class first before the 16 attributes, into a .csv with the output class at the end. In addition, the output class was modified from the letters A-Z to 0-25 as ABAGAIL is better suited to handle numbers. Furthermore, the ABAGAIL code was updated to convert 0-25 categorization into one-hot categorization to remove the bias that implies more similarity between 1 & 2 than 1 & 25 because the numerical difference is smaller.

Using modified code from ABAGAIL's AbaloneTest, it was determined that there was some error in the way the class was comparing the predicted and actual instances. Despite several attempts to correct this, Part 2 was carried out using modified code from MosDragon on Github [3]. In addition to enabling proper comparison of the three random optimization methods, the class also enables cross validation and train/test set splits. For the purpose of this paper, cross validation has not been implemented, although including this would be more comparable to the 10-fold CV used in Assignment 1. However, 10-fold CV adds much more computation, and many insights may be highlighted without

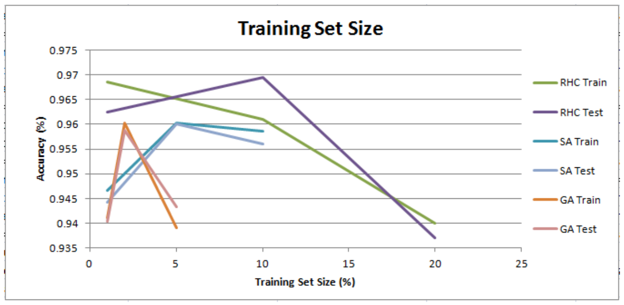
CV. The training set % used was 0.7. The neural network parameters are initialLearningRate = 0.1, maxLearningRate = 50 and minLearningRate = 0.000001.

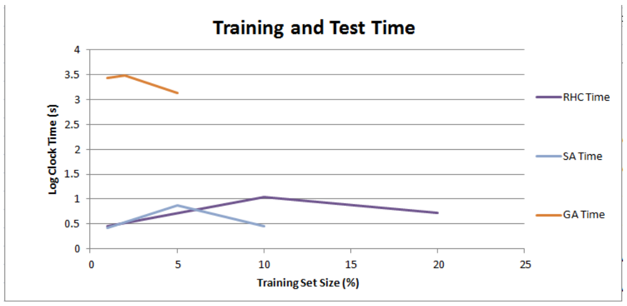
Bias to recognize in this report are preference bias for simple models and those that computer faster to enable a higher volume of iteration while providing information to learn from. Another bias to consider is restrictive bias which is inherent to the dataset which looks only at digital fonts as opposed to handwriting. Another restrictive bias is the way the neural network incrementally changes the weights and random optimization parameters. In the case of the implemented code, an array of acceptable values is populated a priori and the “best” model is determined from the limited list of possible parameter values.

Algorithmic Analysis

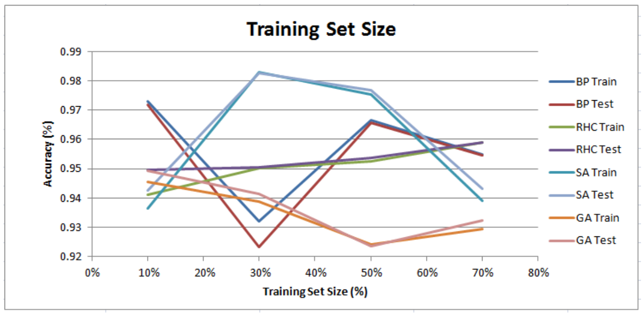
The following sections evaluate the ANN performance when varying the number of iterations, the size of the training set, and type of optimization algorithm, and the parameters used to build the model.

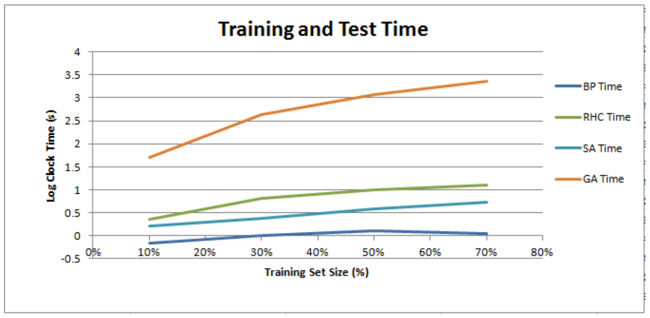
The chart below shows that at very low iterations, the neural network already achieves high accuracy. The accuracy more or less does not improve with more iterations.





The chart below demonstrates that a representative model is generated from the training set to classify the test set. The graph also shows that SA performs well for this problem set, particularly at 30% training size. The right-hand graph shows that SA takes significantly longer to compute than the other methods. As observed in Part 1, typically genetic algorithms and MIMIC are computationally more intensive. Since this example has 16 attributes, the hypothesis space is much more complex and the computations for GA grow exponentially with the attributes as it tries to mate each combination of attributes to determine what traits to mutate.





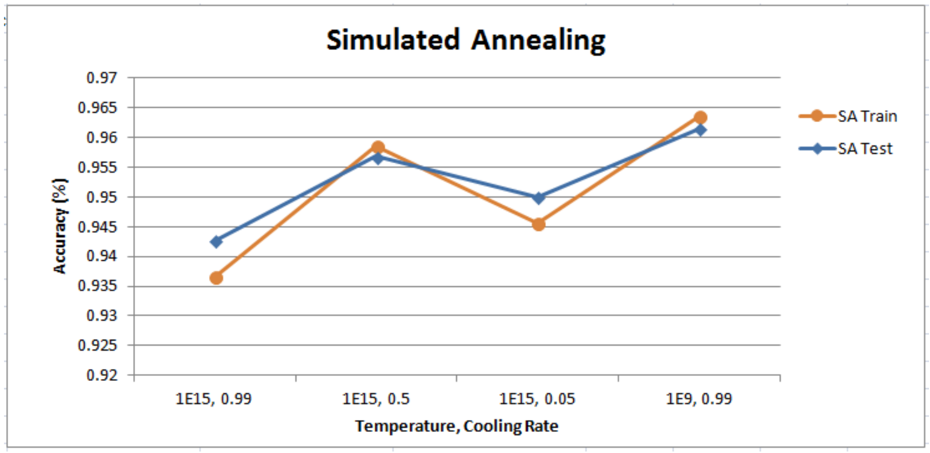
Randomized Hill Climbing

Randomized hill climbing performs quite well for this problem, with about 97% accuracy in 10s training and test time. RHC works well in neural networks because the weights are continuous values and RHC employs a similar principle to gradient descent which tests neighboring points to incrementally climb towards local optima or settle at a peak.

Simulated Annealing

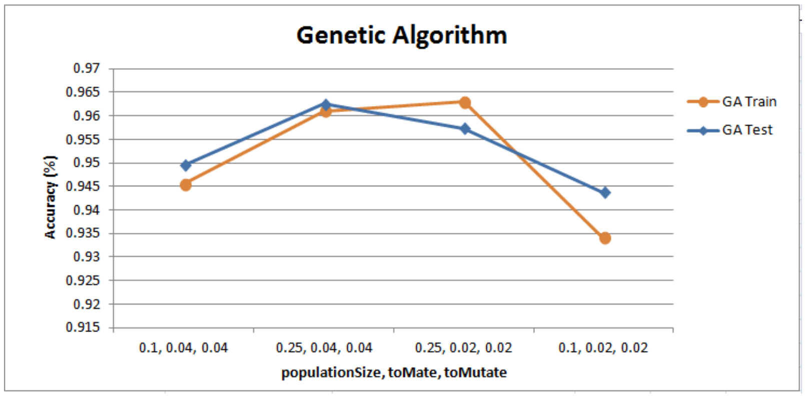
The implemented code was initially run to determine the ideal temperature and cooling rate by modifying it between iterations. The optimal parameters were generally temperature between 1E9 and 1E15, and cooling rate between 0.50 and 099. The parameters 1E9 and 0.99 performed quite well, and were selected for further analysis. The lower temperature and high cooling rate indicate that some exploration is important at the start; however high exploitation is important to select the optimal neural network weights.

This graph demonstrates the high T and high cooling rate do not pair well, and neither do low T and low cooling rate. This results in a model that does not carry the advantage of exploring or exploiting.



Genetic Algorithm

The implemented code was initially run to determine the ideal ratios for population, mate and mutate. The optimal parameters were around 0.25, 0.04 and 0.04. The accuracies seen range between 93-95%, which is considerably high, but slightly less than the RHC performance. In genetic algorithm, mutation provides exploration while cross-over leads the population to converge on good solutions (exploitation). Consequently, there is a balance between cross-over trying to converge and mutation trying to avoid convergence and explore more areas. Unfortunately, beyond a light understanding of how mate and mutate ratios affect exploration and exploitation, GA is known to be a black box for finding optimal solutions while unable to make it clear or intuitive.



Conclusion

Overall, simulated annealing appeared to perform the best at 98%. This suggests that the optima are more distinct (as exploitation is more important that exploration). In comparison to the WEKA analysis, which achieved a maximum 85%, all three algorithms perform better. This, of course, is problem dependent and further study is highly recommended. The learning rate and momentum in the WEKA model may need to be revisited.

Bibliography

[1] https://en.wikipedia.org/wiki/Combinatorial_optimization#Applications

[2] https://github.com/pushkar/ABAGAIL

[3] https://gist.github.com/mosdragon/ad893f877a631260e3e8

[4] Dataset 1: Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml/datasets/Letter+Recognition]. Irvine, CA: University of California, School of Information and Computer Science.