Random Optimization Comparisons

Nirave Kadakia

Random Optimization Algorithms for Neural Network Weighting

Overall Description of Classification

Classification of Red Wine quality was done earlier with Neural Networks using backpropogation. However, there are various other methods that can be used to weight Neural Networks, and this includes Random Hill Climbing, Simulated Annealing, and Genetic Algorithms. The purpose of the first part of the paper is to compare these Random Optimization algorithms in Neural Networks

Overall Description of Data

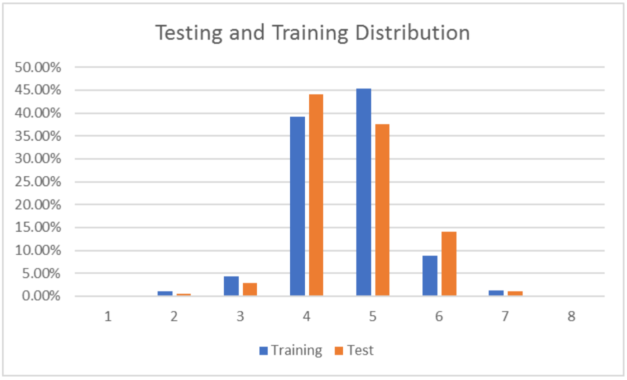
Red Wine quality data, and a complete description of features, is located https://archive.ics.uci.edu/ml/datasets/Wine+Quality. This dataset contains 11 attributes based on physiochemical tests, ranging acidity, pH level, alcohol level and more. This is used to predict one attribute ranging from 3 to 8 describing the quality.

Red Wine quality is important because wine is expensive to procure and choosing the wrong wine can cost a lot of money to both an individual and wine merchants.

Splitting of Data into Training and Validation

Red Wine quality will be split, randomly, with 70% training, and 30% validation.

For both datasets, it is important that the training and validation classification be proportionally similar. Here is a graph of the validation and test to visual show that the proportions of wine quality ratings are roughly equivalent.



Types of Random Optimization Algorithms

Random Hill Climbing is an algorithm that approximates an optimum by hill climbing, essentially climbing, or randomly choosing a state and moving up that state if the optimum improves.

Simulated Annealing is an algorithm that approximates a global optimum (i.e. highest temperature) by progressing to different states through an iterative process. Starting at a random state, it continues until the maximum number of steps have been reached or the error has reached a minimum. It progresses to a next state by picking a neighboring state at random and, if the temperature is higher at that state, it moves. If the temperature is not higher, it will move only based on a certain probability in order to avoid local optimas.

Genetic Algorithm is an algorithm that resembles natural selection through mutation and crossover and selecting the most fit individuals. Essentially, until converged, it computes and finds the most fit individuals over a population, then replaces the least fit with new values via crossover/mutation of the most fit, and uses that as the new population.

Methodology

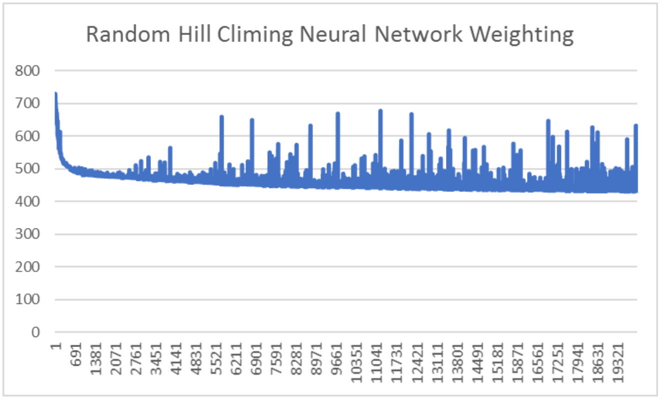
The initial backpropogation algorithm was done using R's nnet library, and the description of the methodology is in the last assignment and will not be repeated.

The Random Optimization algorithms used in weighting the Neural Network was done using ABAGAIL, specifically altering the AlbaloneTest.java file by altering the following:

1. Changing the input file to the red wine csv file
2. Changing the binary options to multi-label classification
3. Changing the hidden layers to 2, with 15 nodes each, and altering the number of iterations accordingly

Random Hill Climbing

With Random Hill Climbing, 20000 iterations were run and the Root Mean Squared Error was plotted after each iteration.

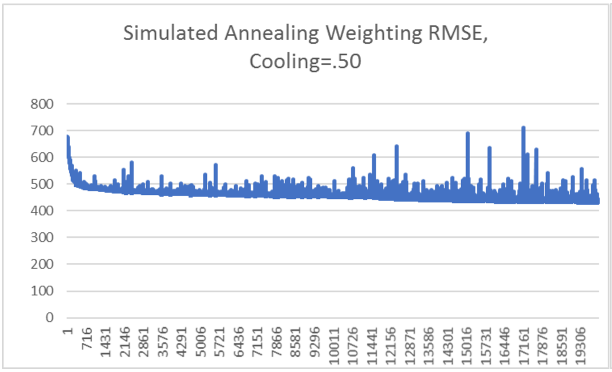


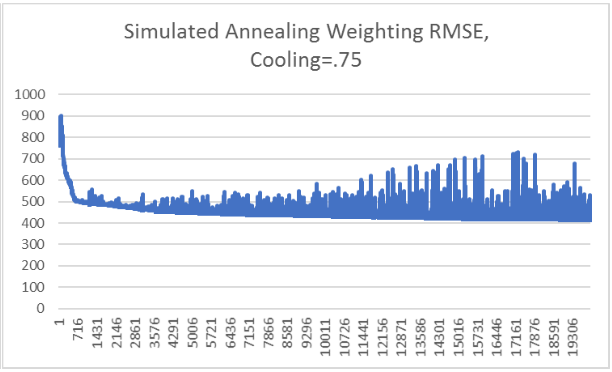
It is clear to see that though there are random bumps in RMSE, which coincide with the randomness that Random Hill Climbing entails, the RMSE goes down until about 15,000 iterations and then approximately levels out.

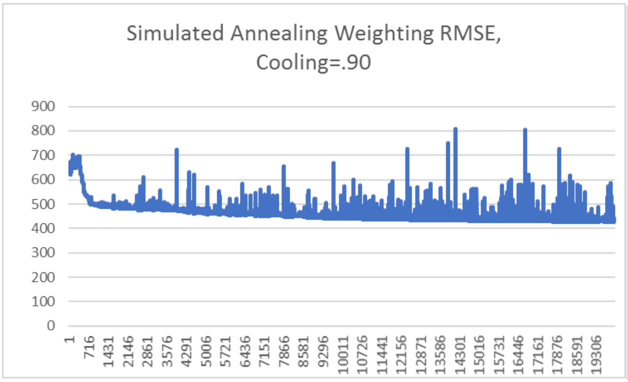
The total time needed was 1059.17 seconds to train, yielding a training accuracy of only 59.8% and testing accuracy of only 47.25%. While predicting a score from scale 3-8 with 47.25% is decent, it does not match the Neural Network with backpropogation in both time (even if the RHC algorithm was reduced down to 15,000 iterations) nor accuracy.

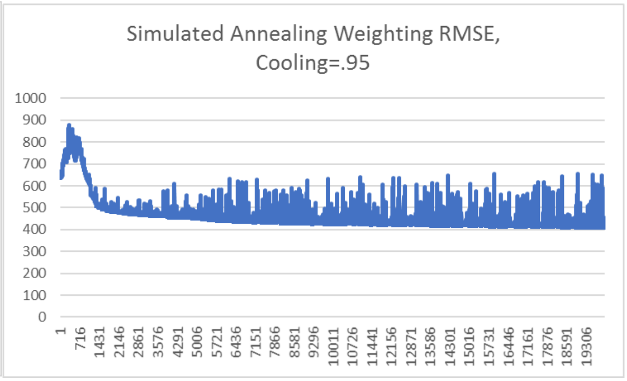
Simulated Annealing

With Simulated Annealing, 20000 iterations were run and the Root Mean Squared Error was plotted after each iteration. This was run with different cooling parameters 0.50, 0.75, 0.9 and 0.95 , and the best testing accuracy value was used.









Though there is an initial bump up (due to the random nature of Simulating Annealing), the trend suggests that s the number of iterations increase, the number of iterations increase as well.

The best testing accuracy occurred with a cooling factor of 0.90. The total time needed was 1065 seconds to train, yielding a training accuracy of only 59.7% and testing accuracy of only 47.5%. Again, it does not match the Neural Network with backpropogation in both time nor accuracy.

Genetic Algorithm

With Genetic Algorithm, the following iterations were run for 100 iterations with the following parameter to obtain the best value for population, keeping the rest of the parameters constant at the default values:

Population	toMate	toMutate	Training Accuracy	Testing Accuracy
100	100	10	42.6%	46.5%

200	100	10	48.1%	48%
300	100	10	37.5%	38.5%
400	100	10	43.7%	49.5%

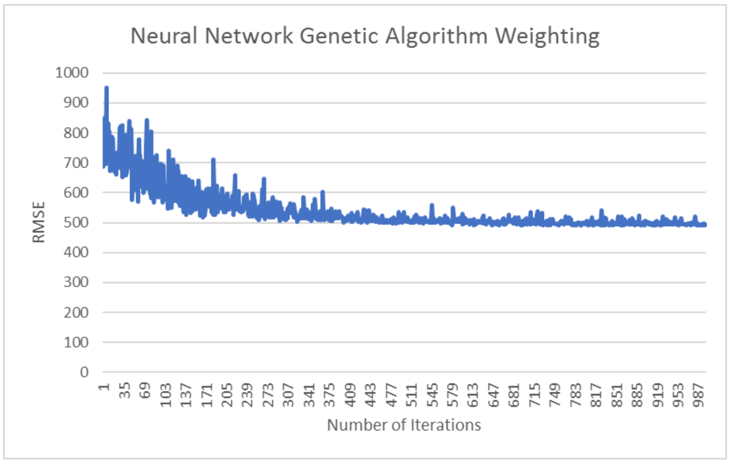
The same was done for toMate with population set to 200 (the best value from above)

Population	toMate	toMutate	Training Accuracy	Testing Accuracy
200	50	10	48.5%	45%
200	75	10	50.7%	42.7%
200	125	10	48.1%	42%
200	150	10	48.8%	46%

The same was done for toMutate with the population set to 200 and toMate set to 100 (which contained the best total error rate out of all from above)

Population	toMate	toMutate	Training Accuracy	Testing Accuracy
200	100	5	41.7%	37.8%
200	100	15	46.5%	42.5&
200	100	20	47.3%	47%
200	100	20	49.2%	44.25%

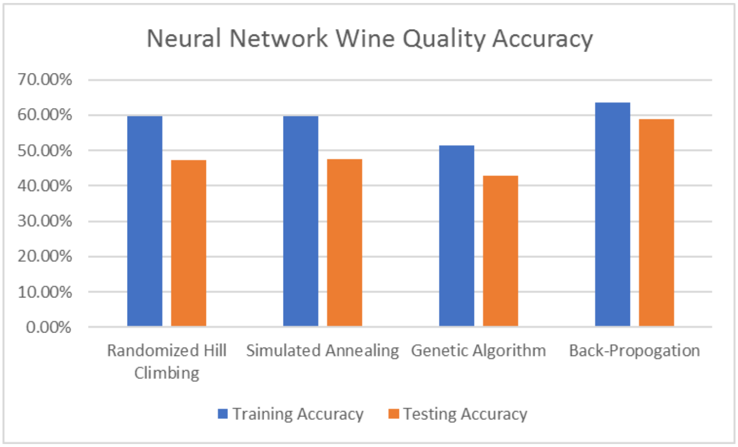
After which, the best parameters, population=200, toMate=100, toMutate=15 were used to run Graduate Algorithms weighting Neural Networks with 1000 iterations, with the Root Mean Squared Plotted.



The final results of 51.26% training error and 42.5% testing error with a training time of 2586 seconds.

Comparison with Backpropogation

It is quite easy to see that backpropogation is the superior neural network weighting algorithm for this particular case. This is most likely due to many factors.



The backpropogation training tends to have better results in this case maybe because backpropogation allows for the nodes to be weighted with continuous, real data. Random optimization algorithms, like Genetic Algorithm, however, does better with discrete values. Perhaps applying discrete changes loses a bit of the fine tuning that is needed for creating a proper neural network. In addition, an algorithm like GA requires proper crossover/mutation operators, which is sometimes difficult to achieve. This is most likely why GA returned the poorest results.

In addition, timing for all 3 Randomized Optimization problems took a long time (> 1000 seconds). Compare with the less than 30 seconds needed for backpropogation. Backpropogation is quicker because the training method alters many weights at once through each iteration, as the errors are "back propagated" from the output all the way thoughout the network. The other algorithms either make minor changes (e.g. find a "neighbor" in SA and RHC), or make slight chances (mutation), which takes longer. Genetic Algorithm., in particular, must create entire populations (e.g. weights), so the running time was the longest at over 2000 seconds.

Conclusion

Random Optimization problems can be used to weight neural networks, though they may not necessarily lead to better results. Results will take much longer due to iterative process of finding an optimization. In other words, the type of problem will dictate the best possible algorithm to use. For the wine data set, backpropogation seems to work best.

Random Optimizations Against 3 Different Problems

The second part of this paper will deal with solving 3 different problems using 4 different Random Optimization algorithms in order to provide a comparison of their respective strengths and weaknesses.

Algorithms

The following algorithms will be evaluated:

Randomized Hill Climbing, Simulated Annealing, Genetic Algorithms

See above for description

MIMIC

MIMIC is a randomization algorithm that idirectly models a probability distribution and successfully refines that model so that along with maximizing the fitness function, it will also convey structure. It does this by sampling uniformly from all the points and refines it until it gets to a distribution of optimal points. The algorithm basically samples, retain only samples that are above a certain percentile, estimates a new probability distribution (which contains the structure and is often represented as a dependency tree), and repeats.

Problems

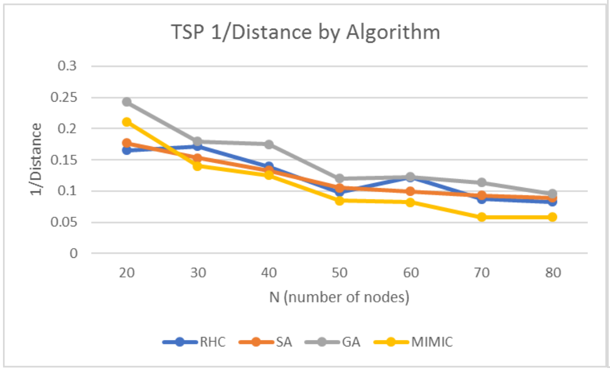
The Random Optimization algorithms above will be run with the following optimization problems to highlight the advantages and disadvantages of each algorithm.

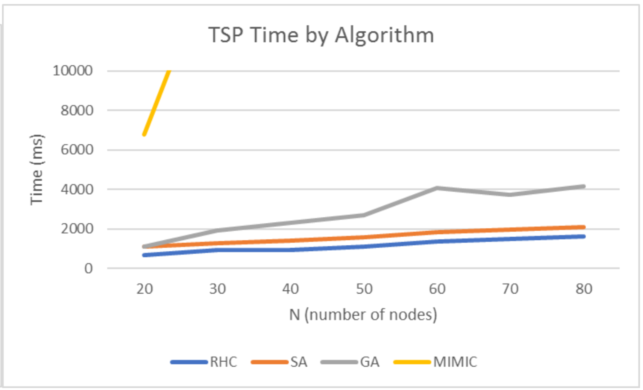
- Traveling Salesman Problem
- Continuous Peaks
- Max K-Coloring

Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is a problem of minimizing the distance a salesman must travel to visit all cities. The generalized version is essentially finding the minimum distance to traverse all nodes within a graph of subgraph, where each edge has a certain distance.

TravelingSalesmanTest.java from ABAGAIL was used with small alterations to print out the times and run the algorithm multiple times through different values of N. All 4 algorithms were run with differing values of N. Below is a graph detailing the 1/distance (meaning higher is better) and the number of nodes. In addition, a graph of the times is shown as well. For the time graph below, note that MIMIC grows exponentially (80 nodes is 256 seconds) and is not displayed on this graph beyond N > 30.





Results:

While all algorithms perform reasonable well in terms of distance, Genetic Algorithm tends to returns the best result, followed by Simulated Annealing and Randomized Hill Climbing. This makes sense as

TSP tends to lend well with slight mutations. In other words, finding a series of good paths (a population), and mutating them to create a better overall series of paths (a better population), and repeating, is a good way to solve the problem. Intuitively speaking, when a person solves problem, a series of routes is usually suggested, and combinations/changes to these routes to get better routes would yield a good result (mimicking GA), as opposed to just picking one route at random and just making a change once as you go (mimicking RHC and SA)

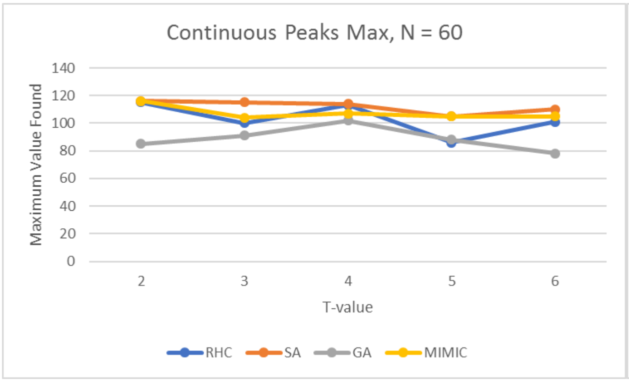
In terms of time, as the population grows (e.g. number of nodes), RHC and SA seem to grow logarithmically, meaning that extra nodes do not pose much of a problem. GA seems to grow quite a bit since the population to choose from and mutate grows as well. MIMIC grows exponentially, and is a poor choice for solving this problem, as the construction of dependency trees (and thus the time for each iteration), grows larger as N grows, as trees grow exponentially.

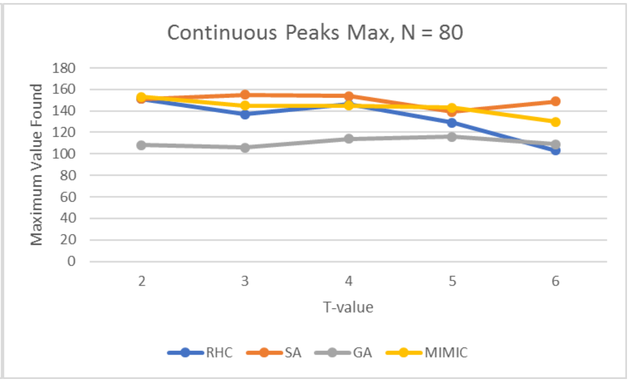
Continuous Peaks

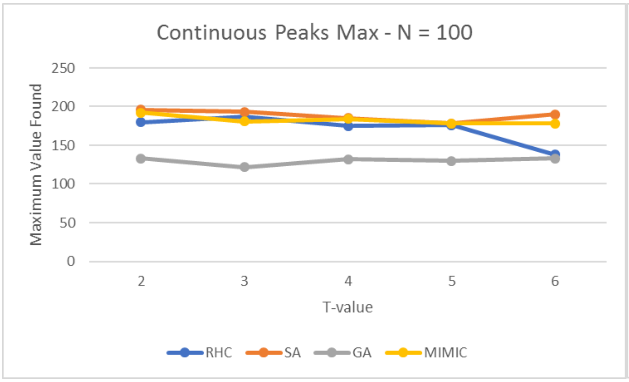
The Continuous Peaks problem is where, given a random set of integers with peaks and valleys (defined by basins of attraction), what is the maximum value? Essentially, given an XY graph, what is the largest Y?

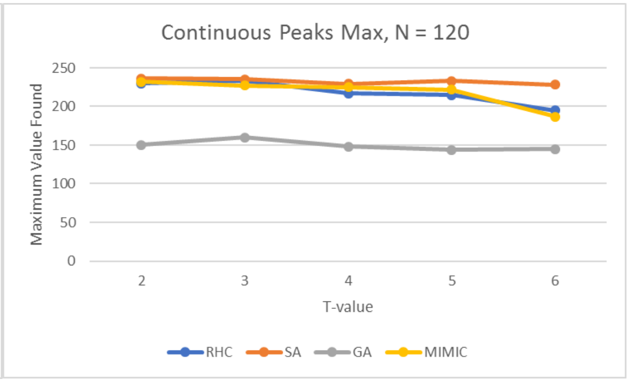
There were multiple problems run. This was tried with various N (the number of integers) and T (the basin of attraction). Below are graphs of N with various T values.

Below are the optimal values found given 4 different N values and T values ranging from 2-6 across all algorithms:

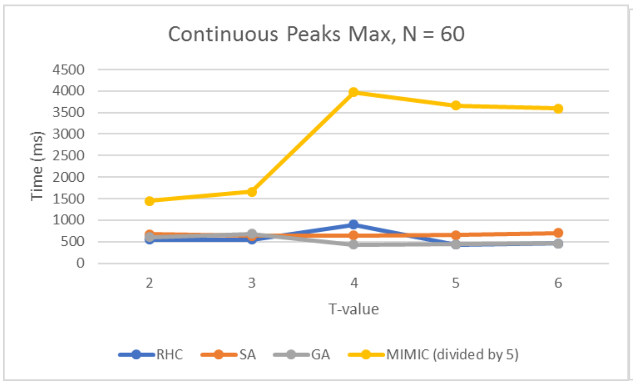


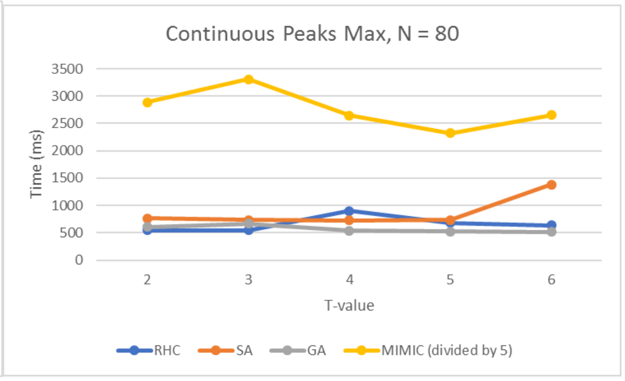


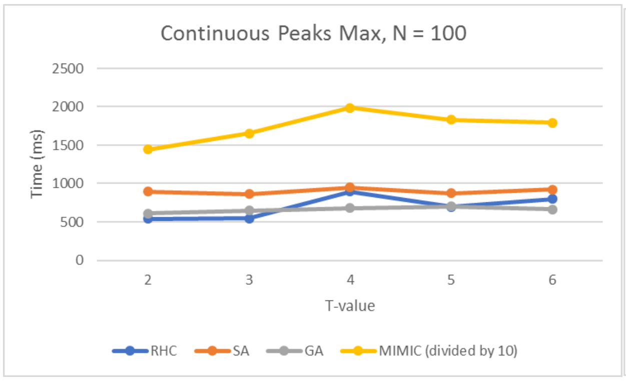


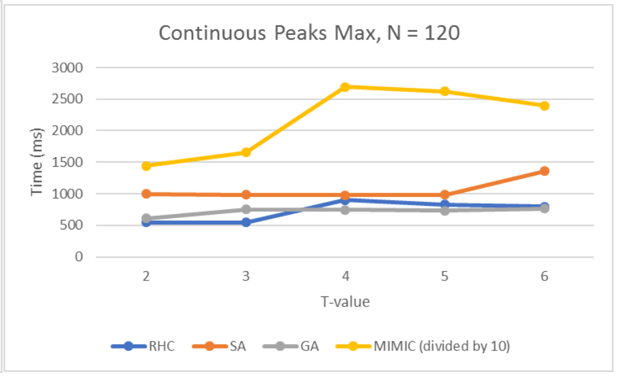


And below are the times for those same runs – note that MIMIC's times are divided by 10 in order to fit on the graph.









Results:

It is clear to see that Simulated Annealing yields generally the best results. This makes intuitive sense because Simulated Annealing is made to climb hills, which is what finding the highest Y value on an XY graph is.

With 2 being low basin of attraction, there are not many local optima, which shows that Randomized Hill Climbing and Simulated Annealing yield similar results when t is low. As the basin of attraction increases, however, so does the local optima, and Simulated Annealing, which is made to avoid local optima, yields vastly superior results.

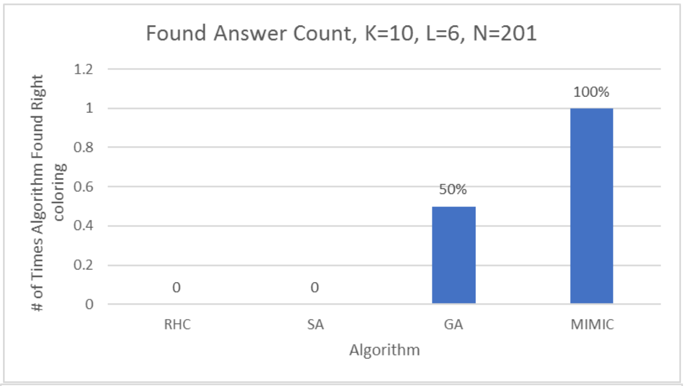
Continuous Peaks, on the other hand, does not work well with Genetic Algorithms, as the problem itself does not lend itself well to mutation and creating multiple populations is not that advantageous. In other words, combining two high values (i.e. mutation/crossover) would not necessarily lead to a good new point. For example, if two points near two local optima are crossed over, the child point could be in a valley, and thus a low value is introduced into the population).

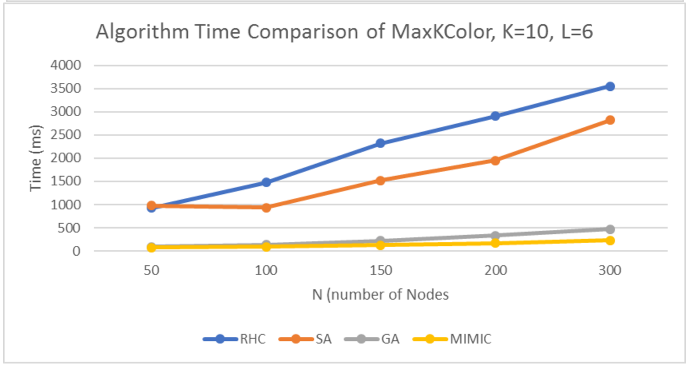
Similarly, time for the results shows that Random Hill Climbing being the quickest, that is because the algorithm is geared towards climbing hills (which is what Continuous Peaks is doing), but is much quicker because it may get stuck with a local optimum and quit early. Simulated Annealing is quick, but the exploration aspect of SA takes a bit longer. The outlier is MIMIC, which is extremely slow. MIMIC works well when the evaluation function takes a long time, but the number of iterations is not large. However, Continuous peaks requires lots of iterations, and MIMIC's construction of trees slows it down considerably.

MaxKColor

The Max k-coloring problem is where, given a node colored graph of K number of colors (i.e. each node has a color), is there a coloring that allows no same colored adjacent node for all nodes?

MaxKColoringTest.java from ABAGAIL was used with small alterations to print out the times and run the algorithm multiple times. To test accuracy, the algorithm was run with K=10 and 201 nodes (N), and 6 adjacent nodes per node (L) 10 times to find out if the algorithm could find the appropriate answer. In addition, time to run was analyzed over differing levels of nodes, with L set to 6 and K set to 10. These times were for both failed and successful algorithms.





You can see that MIMIC not only returns the best results, but also the fastest results. This is due to the fact that the MaxKColor relies on state and structure, which MIMIC is known for keeping. MIMIC, in fact, was the only one to solve all 10 problems with K=10, L=6, N=201. Since this problem relies on the state of other nodes, MIMIC is a natural fit. Randomized Hill Climbing and Simulated Annealing, however, were unable to solve this problem at all. This is because a slight change in one variable could nullify all the gains made. In other words, one small change "close by" does not yield incremental improvements.

In terms of time, GA and MIMIC are both low as they rely on incremental changes/mutations as opposed to jumping to a new coloring. MIMIC being the lowest because it, again, relies on state so that it can quickly get to the answer with just a few iterations.   Randomized Hill Climbing and Simulated Annealing took the longest and seemed to grow linearly, as they continued an exhaustive approach proportional to the number of nodes and did not stop early as no answer was found.

Conclusion

It is clear that certain algorithms work well for certain problems. All algorithms have strengths and weaknesses as witnessed with the 3 problems above. Randomized Hill Climbing works well for problems with incremental improvements and lots of iterations, but do not have local optimum, and there is one optimum (e.g. one maximum value like in Continuous Peaks). Simulated Annealing takes slightly longer, but solves the local optima problem. Genetic Algorithms work well when mutation is in play with an entire population (e.g. a graph), and MIMIC works well when state is a consideration (e.g. the neighboring nodes' color).

References:

ABAGAIL: http://stackoverflow.com/questions/20370827/plot-learning-curves-with-caret-package-and-r/40885119#40885119

Simulated Annealing Wiki: https://en.wikipedia.org/wiki/Simulated_annealing

MIMIC: http://www.cc.gatech.edu/~isbell/tutorials/mimic-tutorial.pdf

Stack Exchange about RHC and Neural Network: http://stats.stackexchange.com/questions/136724/using-randomized-search-algorithms-to-find-weights-for-neural-network

Genetic Algorithm versus Simulated Annealing--performance comparison and use cases - http://stackoverflow.com/questions/4092774/genetic-algorithm-versus-simulated-annealing-performance-comparison-and-use-cas