CS 7641 - Randomized Optimization

Author: Swapnil Ralhan(sralhan3@gatech.edu)
Date: 10/19/2014

Overview

The purpose of this paper is to analyze the performance four local random search algorithms:

1. Randomized Hill Climbing
2. Simulated Annealing
3. A Genetic Algorithm
4. MIMIC [1]

We explore the first three algorithms on generating weights for a neural network for the problem of generating classifying letter image recognition into the 26 alphabet categories.
The second part of this paper explores 3 different problems over discrete-valued parameter spaces:

1. Four Peaks
2. Knapsack
3. Travelling Salesman

The problems are described in detail in a later section. We compare the performance across different metrics of these 3 problems on the later 3 algorithms listed above - Simulated Annealing, a genetic algorithm and MIMIC.

Analysis Method

For the purposes of these problems, we use a different analysis method than used in other papers(such as MIMIC[1]). Instead of running the experiment for a fixed number of iterations, we run it over a fixed time. After every iteration of training, we test the elapsed time to the total time allowed, and proceed if the total time elapsed has not exceed the allowed time.(This does cause some caveats, where the something with a very large per-iteration time, such as MIMIC, or a Genetic algorithm with a large population size would end up getting an advantage). In general, we choose this method as it is contrasted with the per-iteration tests presented in other analyses, and gives another view, which we believe is fairer, as a user of the algorithm is concerned with how long the analysis takes, not necessarily the number of iterations if the time taken by each iteration differs vastly. Though this method might be susceptible to the specifics of the machine being run on, the general relative results obtained(rather than the specific number) should still stand.

Neural Network

Problem - Letter Image Recognition

Description

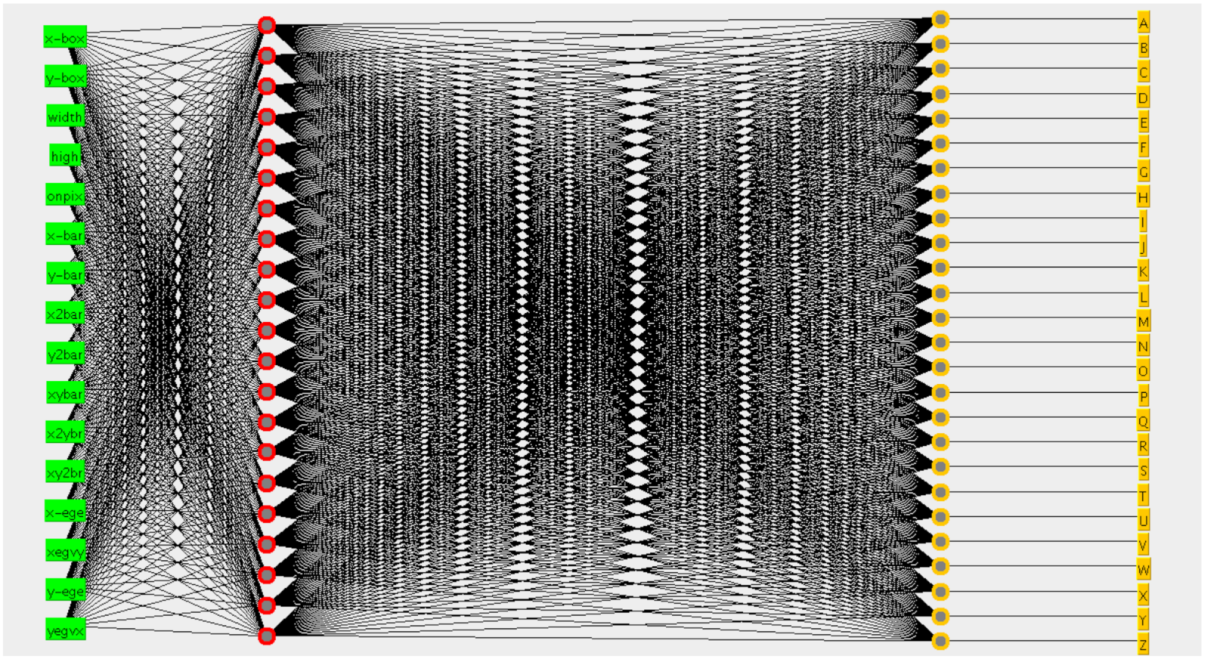
The objective is to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet. The character images were based on 20 different fonts and each letter within these 20 fonts was randomly distorted to produce a file of

20,000 unique stimuli. Each stimulus was converted into 16 primitive numerical attributes (statistical moments and edge counts) which were then scaled to fit into a range of integer values from 0 through 15.

Number of Instances
20000

Number of Attributes
16 (numeric features)

Number of Classes
26 capital letter (26 values from A to Z)

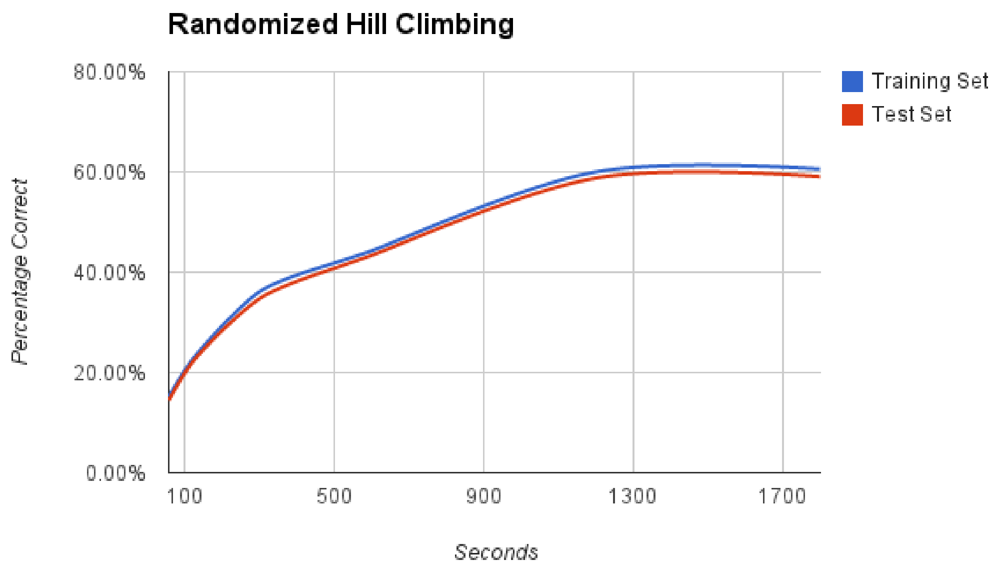


We use 16000 instances for training and 4000 instances for testing. The neural network generated as part of the previous round of testing(assignment 1) is demonstrated above. It has 2 hidden layers, the first with 31 nodes, fully connected to the input layer(with 16 nodes, 1 for each input). The first layer connects to the second layer of 26 nodes, which is one to one connected to the output layer which has 26 nodes for each of the 26 classes. This gives us a total of 1328 links, and the randomized optimization must find the weights for these 1328 links.

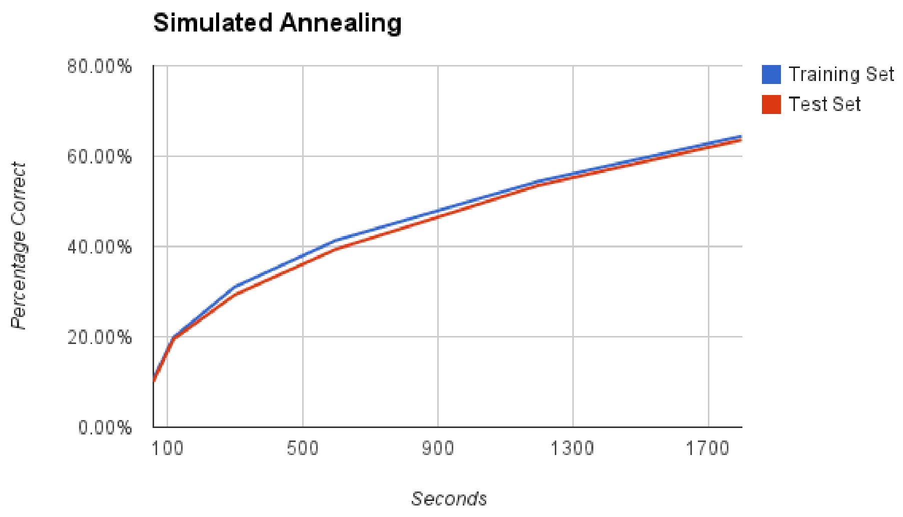
Randomized Hill Climbing

Randomized hill climbing trains slow to begin with, seeing close to 20% correct classification after 100 seconds of training. We see a slow improvement as training time increases, to about 60%, at which point the algorithm levels off.

The slow initial start can be attributed to the need to guess a lot of different weights. We see improvement as we let it train for more iterations as it keeps guessing the lower weights, which are easier to guess. At a certain point, it is hard for randomized hill climbing to guess the accurate weights, as it appears to keep getting stuck in a specific set of local optima, unable to find it’s way out to the global optima.The test set error rate and the training set error rate track very closely, as we would expect. The difference in the test and training set error rate remains fairly constant, not deviating more and more as one would expect in the case of overfitting.



Simulated Annealing



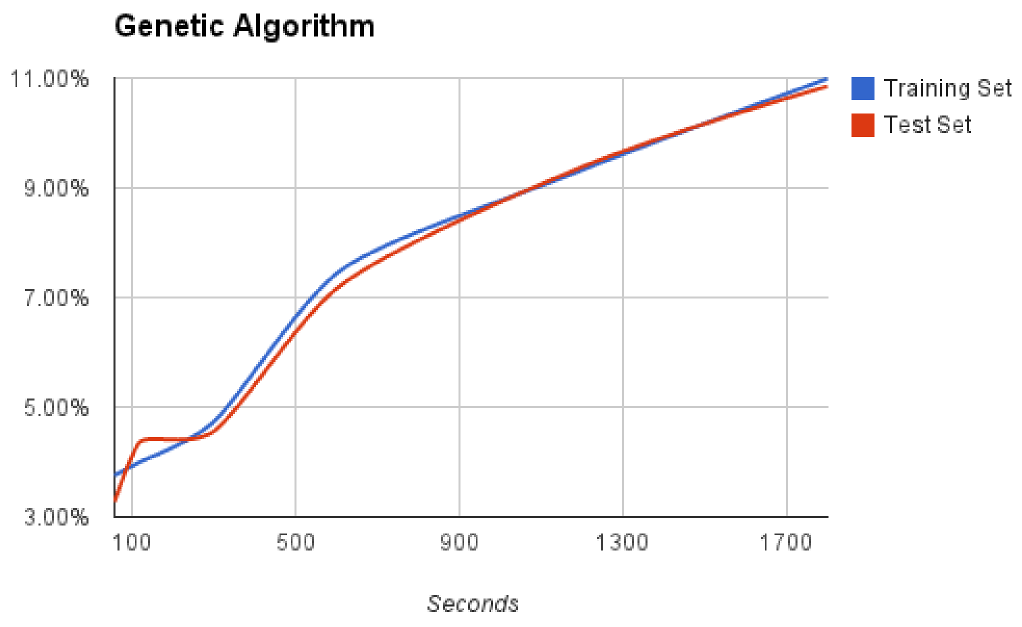
Simulated Annealing has a similar start as Randomized Hill Climbing, starting with a very low classification rate. Similarly, the performance steadily improves, going past 60% after 30 minutes of training.

Given the similar natures of Randomized Hill Climbing and Simulated Annealing(specifically the initial random guessing), the initial slow start and slow improvement is expected. Simulated annealing though has the advantage that when making a random guess, the probability of moving into that guess is also proportional to how much is the drop in the output value, thus the guessing gets more conservative over time, and we see steady improvements even past 30 minutes. Again, the test set error rate and the training set error rate track very closely, as we would expect. The difference in the test and training set error rate remains fairly constant, not deviating more and more as one would expect in the case of overfitting.

Genetic Algorithm

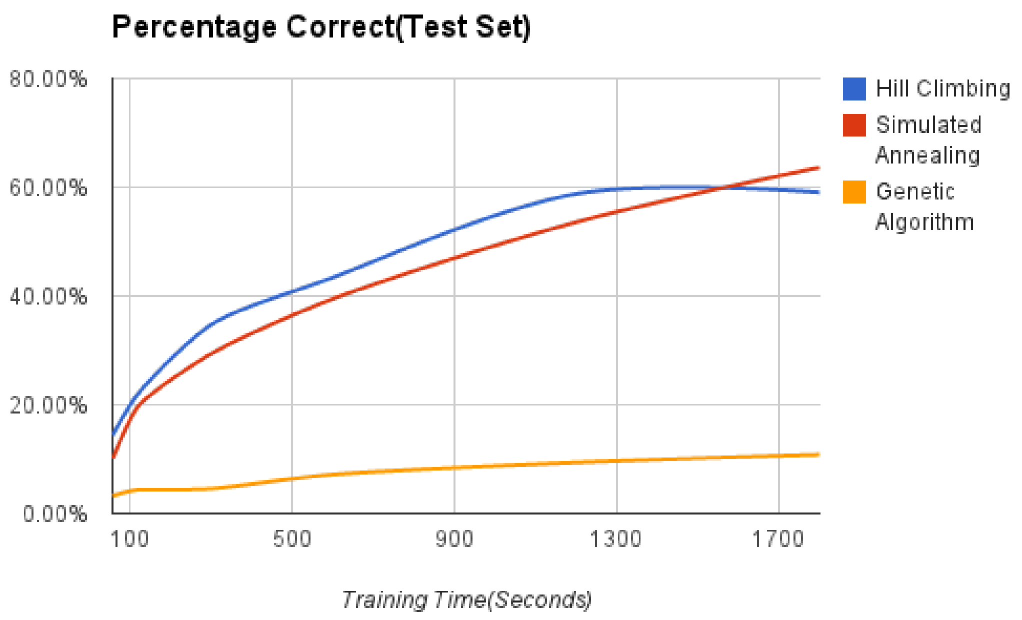
The genetic algorithm has a very slow start, staying at <4% correct rate up till about 300 seconds of training(randomly guessing a class would 1/26 ~ 3.85%). It sees slow improvement, going to about 11% error rate after 30 minutes of execution.

Such a low error rate seems very unexpected, but initially, the algorithm starts with a random 1328 length vector as it's set of weights. Moving from there, it tries to mate the "fittest" population, which at this point is pretty random, so 2 bads simply make another bad. Thus it is unable to make much of an improvement. Furthermore, each iteration takes a lot longer, thus in the given time, we are able to get much fewer iterations of this algorithm.



One metric that we played with is the population size, population to mate and population to mutate. Keeping the population to mate and population to mutate as a fixed percentage of the population size(50% and 10% respectively), we see that training accuracy improves with population size, but so does training time per iteration(as we would expect, more samples to take per iteration). Thus we picked a population size of 2000, roughly double the number of metrics to guess.

Comparison



We see a number of interesting results. For one, the genetic algorithm performs much worse than randomized hill climbing and simulated annealing. This is because we are concerned here with performance in a certain time. Since the genetic algorithm does more work per iteration, it's initial performance is slow, as in the given amount of time it can only do a limited amount of random guessing.

The second interesting thing we see is the performance of simulated annealing vs randomized hill climbing. Initially randomized hill climbing performs better than simulated annealing, but after some time RHC levels off, while SA keeps improving, eventually surpassing the accuracy of RHC. This can be explained as follows - initially there is a lot more random guessing by RHC, thus it can find a better value in a limited time. SA tapers off it's random guessing, so it is not as successful initially. On being given a longer time, SA, through it's slightly smarter guessing(guessing inversely proportional to how much worse the new position would be), yields an advantage over RHC, leading to better results.

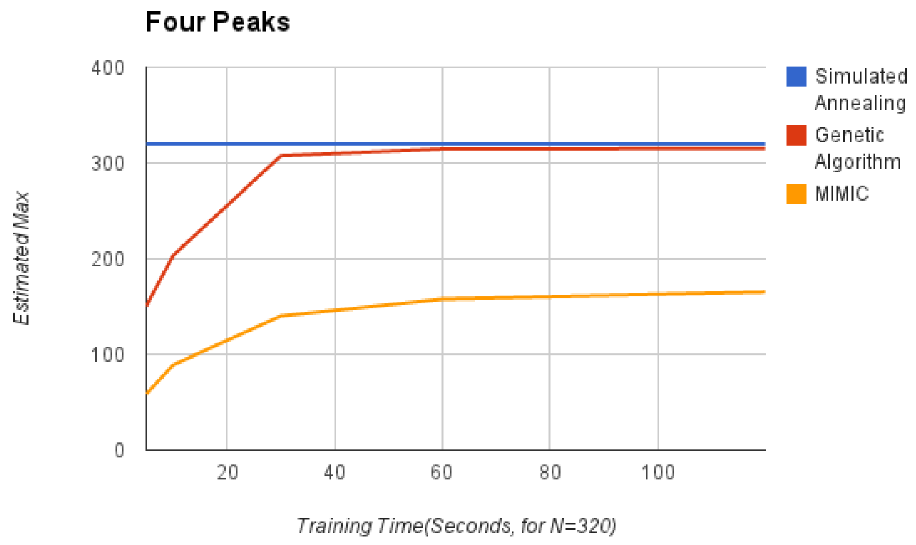
Problems

Problem 1 - Four Peaks

Problem Description

The first problem we look at is the Four Peaks problem [2]. For detailed description, please see [1] Section 5.1

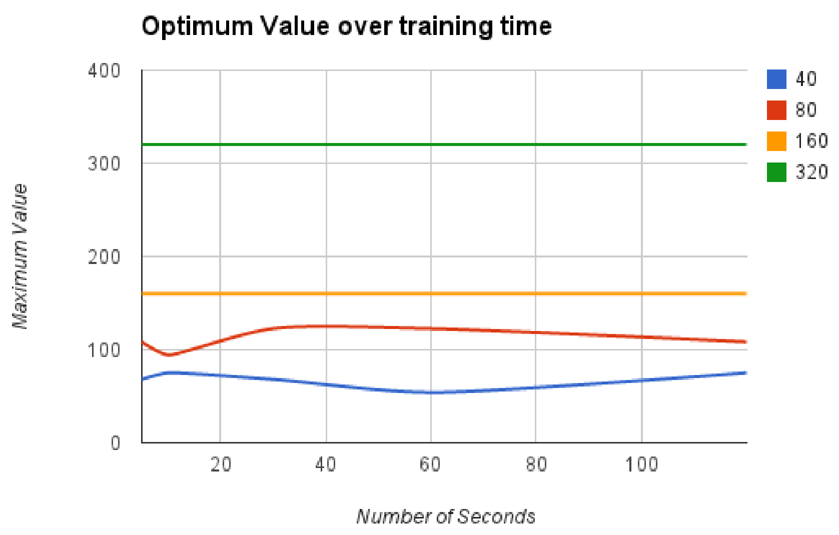
Comparison



We use array size 320 for testing, and compare at different training times(in seconds). We repeat the experiment multiple times and take the average.

We see that simulated annealing quickly discovers the max, starting at a low training time(~5 seconds). The genetic algorithm discovers a lower optima at smaller training times, but after about 30 seconds consistently finds the same optima as simulated annealing. MIMIC on the other hand is unable the find the optima after 160 seconds(the same optima consistently found by Simulated annealing in 5 seconds).

We should note that genetic algorithms and MIMIC, having slower per iteration times, end up getting fewer iterations for the given time period. This however is an expected effect of our testing methodology. Thus we see that for a simple problem, with 2 local and 2 global optima, Simulated Annealing shines, as it can discover the optima much faster than the other algorithms which try to discover and exploit a structure in the data.



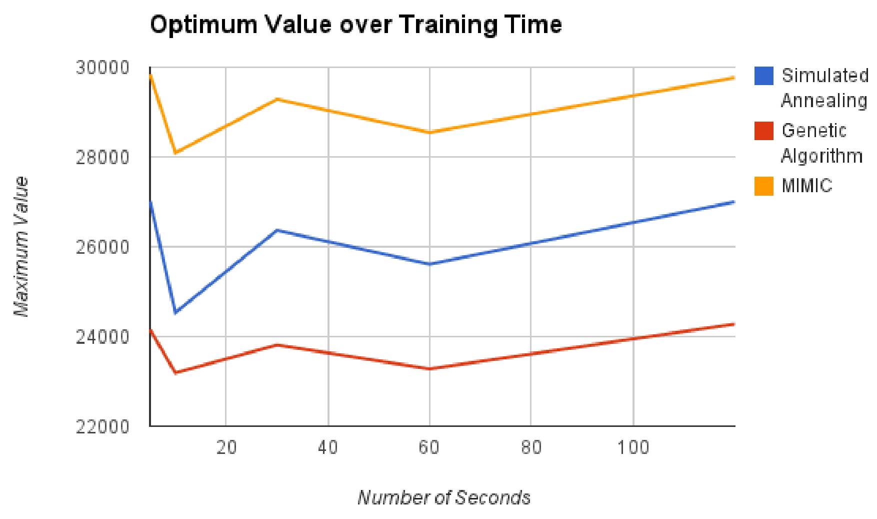
Above we present the results for Simulated annealing over different values of N. As we can see that the maximum value increases with N, and there is not much gained by simulated annealing for any given N as we give it more training time.

Problem 2 - Knapsack

Problem Description

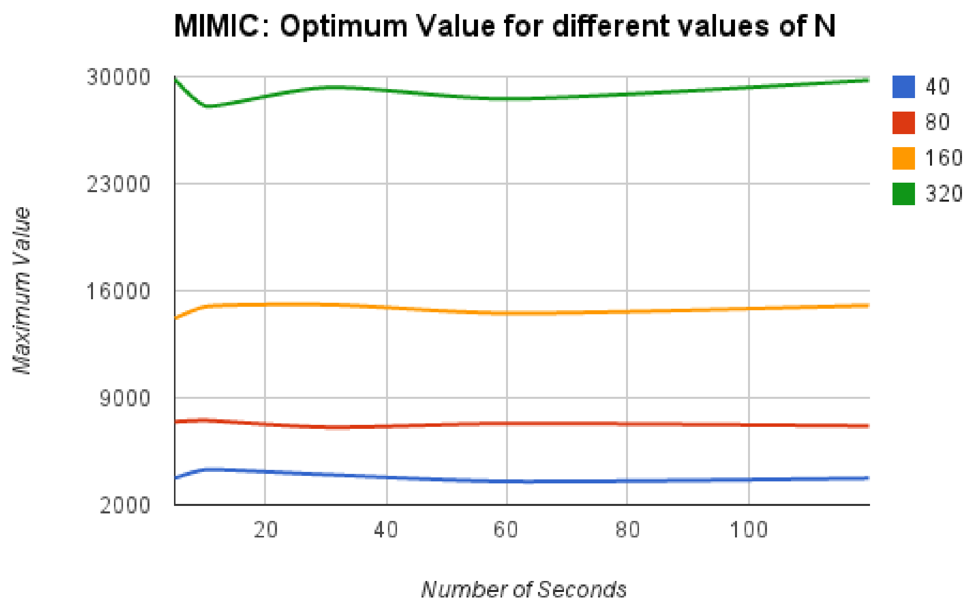
We tackle finding an optimization based solution to the traditional NP-complete Knapsack problem. The problem can be described as follows: Given a set of objects each with a value v_i and a weight w_i, and a weight limit W, set x_i to 0 or 1 to give $max\ \Sigma v_i x_i$ with $\Sigma w_i x_i \leq W$

Comparison



We use 320 items for testing, and compare at different training times(in seconds). We repeat the experiment multiple times and take the average.

As we can see, MIMIC shines in this example, with simulated annealing coming in second and genetic algorithm last. Picking an item affects whether we can pick another item. MIMIC is able to adapt to this structure, and thus can get a better optimal value. The genetic algorithm on the other hand, relies on mating parts of 2 different solutions. This is difficult as picking an item in the first set might exclude some item from the 2nd set, which on combination leads to an unacceptable solution. Thus simulated annealing, which randomly tries out different values, does better than the genetic algorithm.



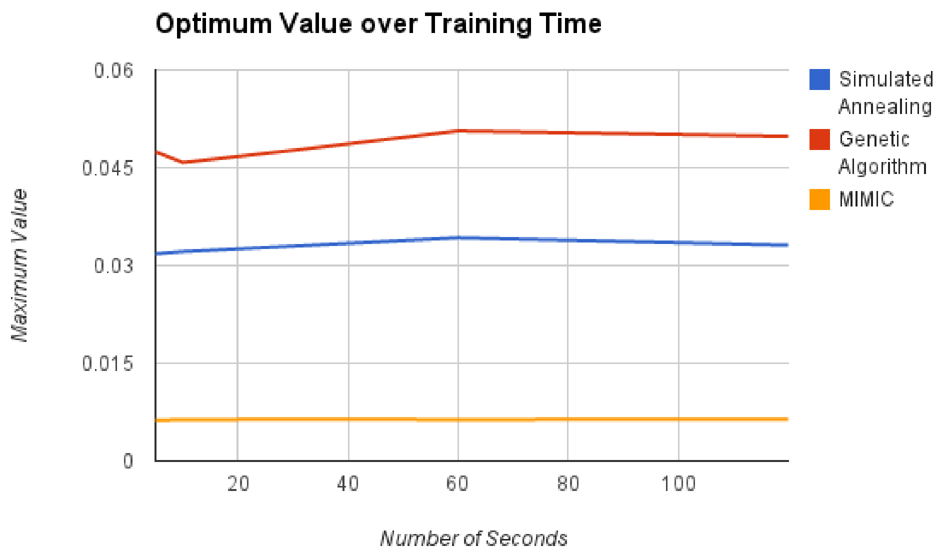
Above we see the performance of MIMIC over multiple different training times for different number of initial items. Comparing to other algorithms(graph not shown), shows that MIMIC outperforms the other 2 algorithms in all cases, and giving it more training time does not affect the optimal value much.

Problem 3 - Travelling Salesman

Problem Description

Last, we look at the classic NP-complete Travelling Salesman Problem. We can define the problem given a Graph = (V,E), a set of vertices and a set of edges, to generate a permutation of $v_i \in V \ s.t. \ (v_k, v_{k+1}) \in E$ to minimize the total distance of the edges selected.

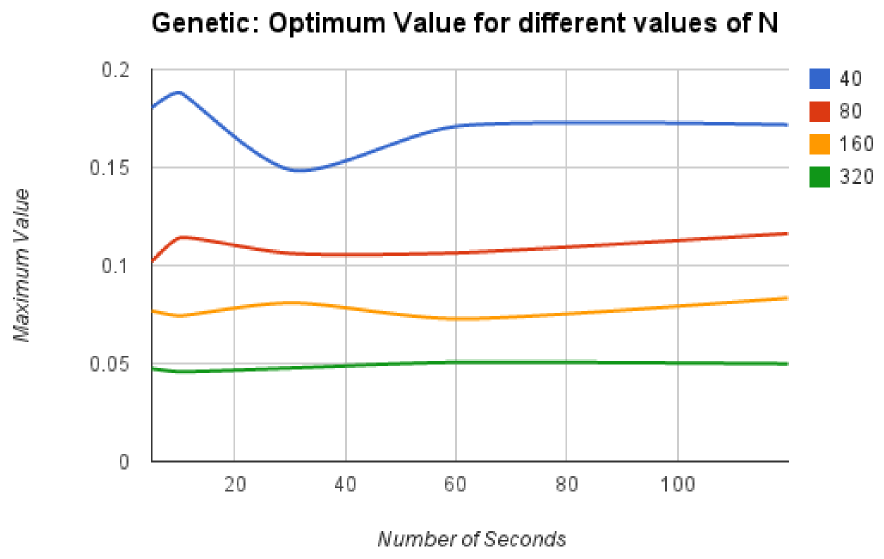
Comparison



We use 320 points for testing, and compare at different training times(in seconds). We repeat the experiment multiple times and take the average. We use 1/distance as the metric to maximize.

The genetic algorithm performs best, with Simulated Annealing coming in second, and MIMIC a distant third. Given the structure of the problem, selecting a set of vertices for the first half of the graph does not affect the set of vertices of the second half too much. Thus we see that genetic algorithm does well, as it combines the good qualities of some “fit individual” with the a later part of the path from another good path. MIMIC does worst because given the limited iteration time, it can only do a few iterations, which leads to the bad behavior.

We also show the performance to the genetic algorithm across different item set sizes. As we can see the distance generated by the TSP is obviously higher(longer path), but comparing to the other algorithms(not shown in graph), it does better across different problem set sizes and times. We also note that the performance does not differ much as we give it more training time, only increasing slightly from T=5 to T=320.



Conclusion

We have presented the four randomized optimization different algorithms - Randomized Hill Climbing, Simulated Annealing, Genetic Algorithm and MIMIC, and their applications to 4 different problem spaces - Image Recognition with Neural Networks, Four Peaks, Knapsack problem and Travelling Salesman Problem. As we have seen, each algorithm has a problem space where it works best. For each of the four problems, we had a different algorithm that gave the best result.

Exploring different problem spaces, it is important to understand each algorithm and how it applies to the problem space. Additionally, there will always be scope for tuning the algorithm to the problem at hand. It is important to try out the algorithm for a number of different input sizes as well as optimization times to understand which is the best applicable algorithm.

There is always scope for more exploration. For instance, we could apply randomized optimization to get the different starting values for each algorithm(initial temperature and cooling for Simulated Annealing, Population Size, Mutation and Mating percentages for the Genetic Algorithm, Samples to take and samples to keep for MIMIC etc). For the purpose of this analysis, we used some domain knowledge combined with a few experiments to estimate rough acceptable values for those variables, but a more thorough approach could be used, especially when in the final stages of optimizing for production use.

References

[1] Jeremy De Bonet, Charles L. Isbell, and Paul Viola. MIMIC: Finding Optima by Estimating Probability Densities. In Advances in Neural Information Processing Systems (NIPS), 1997
[2] S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. Technical report, Carnegie Mellon University, May 1995