Joao Francolin

CS 4641

March 3, 2019

Randomized Optimization

Introduction

The purpose of the report is to explore a variety of random search algorithms. Therefore, the following four algorithms were implemented for careful investigations of their behavior. They are: randomized hill climbing, simulated annealing, MIMIC, and a genetic algorithm. In the next couple of pages they will be explained in two major context:

1) They will be used to train a neural network and subsequently perform a classification problem on political affiliation.
2) They will implemented in three additional problems each, in order to highlight their strengths and weakness.

Background

The Neural Network that will be describe on item 1 was implemented on ABAGAIL. Its training and testing were all done in Java. The problems that will be describe in item 2 ware was implemented on ABAGAIL, and they relied on Jython throughout training and testing. A brief discussion of the four underlining algorithms will be important in order to understand the analyses of the data collected.

Randomized Hill Climbing. RHC employs a gradient decent in order to continually find neighbors with higher evaluation function, and therefore tends to a local maxima. Its weights are continuous, so we expect RHC to perform well finding weights for our Neural Network.

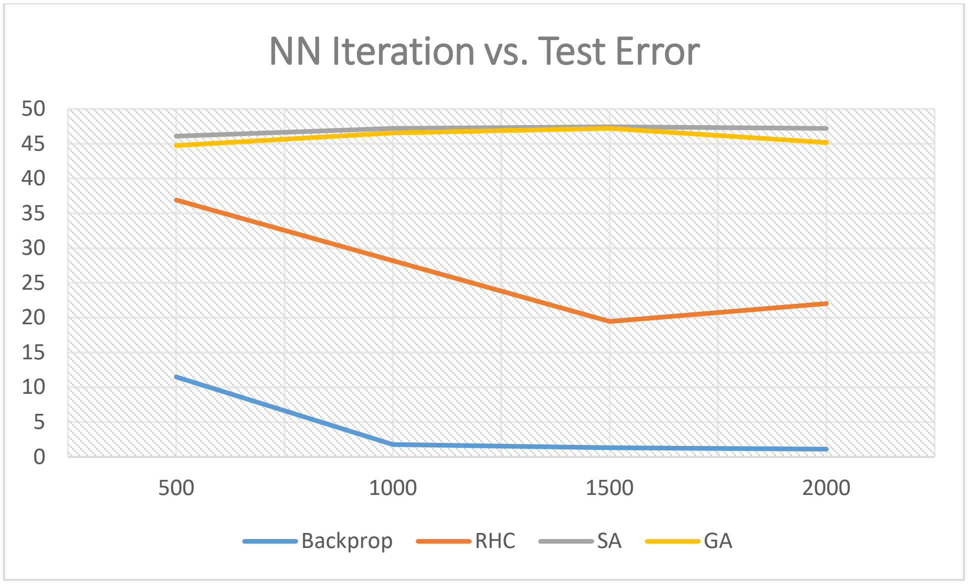
Simulated Annealing. This algorithm relies on two major parameters: a heating rate and a culling rate. These parameters affect how likely the algorithm is to move from its current value; meaning that higher heating rates and lower culling rates will yield on more “exploratory” behavior.

Genetic Algorithm. The Genetic Algorithm uses parameters such as population, mate, and mutate in order to model a balance between an “exploratory” behaviors, seeing heavily on when implementation with high mutation rates, and an “exploitation” behavior, seeing in implementation with high matting rates, and characterizing a convergence patter for a specific solution.

MIMIC. This algorithm relies heavily on probability density functions to find optimal solutions on the search space. It is worth noting that MIMIC is the most computationally heavy algorithm from the four we are exploring.

Part 1: Train and Analyze Neural Network

While studying Supervised Learning and Classification, I made use of sklearn libraries in order to train a Neural Network to classify individual’s political affiliation based on their tweeter feeds. Expanding upon that problem, I used the same tweeter dataset in order to train and test a new Neural Network. I collected 559 tweets feeds that I extract from members of the Senate and House. I created a dictionary, vectorize it, split it into test (20%) and training sets (80%), and finally partitioned the training set into 5 for future cross-validation. I've chosen to use 8 hidden layers, each one 30 nodes wide. The first layer is 300 nodes wide, while the last layer has 2 nodes. The distance function chosen was sum of squared error, and the neural network was trained and tested across multiple iterations. Below is a graph of test error vs. number of iterations for each trained neural network.



The backpropagation method did performed really well, converging to an optimum as soon as 1000 iterations, and having virtually no error on the testing data. Simulated Annealing and Genetic Algorithm in the other hand showed the worst performance, with testing errors doing slightly better than pure chance. This may be an issue with their hipper parameters, but it is worth noticing that they are the ones that took most time to train, and therefore adjusting their parameters for a more exploratory behavior may be unsalable.

```
====Genetic Algorithms====
Number of iterations: 2000
Fold: 1 Error: 46.067416%
Fold: 2 Error: 48.314607%
Fold: 3 Error: 43.820225%
Fold: 4 Error: 51.685393%
Fold: 5 Error: 45.977011%

Min Validation Error: 43.82%
Training Error: 45.76%
Test Error: 45.19%
Time Elapsed: 747.22 s
```

```
=Randomized Hill Climbing=
Number of iterations: 2000
Fold: 1 Error: 17.977528%
Fold: 2 Error: 31.460674%
Fold: 3 Error: 26.966292%
Fold: 4 Error: 19.101124%
Fold: 5 Error: 28.735632%

Min Validation Error: 17.97%
Training Error: 22.03%
Test Error: 21.029083%
Time Elapsed: 219.373 s
```

Notice the discrepancy between times and accuracies from the Genetic algorithm and the Randomized Hill Chilling. While adjusting hyper parameters must eventually make the Genetic Algorithm perform better than chance, it is also possible that RHC is simply a better fir for problem such as binary classification (Democrat or Republican).

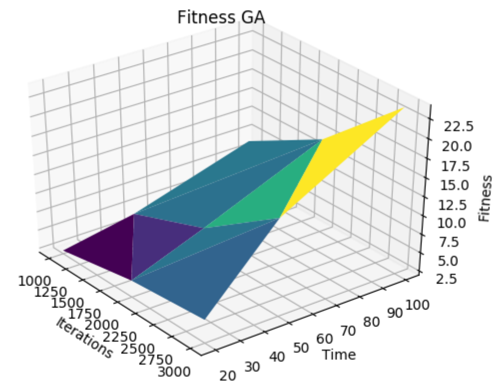
Part 2: Problem Proposal & Algorithm Analyzes

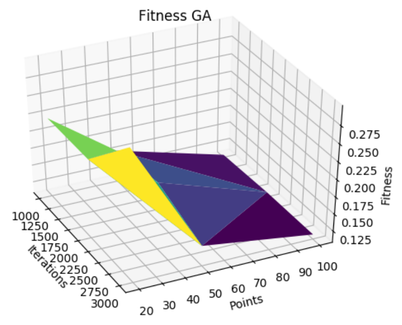
Problem: Travelling Salesman.

This problem consists of a salesman that has to travel to N points and wish to do so in the most efficient manner possible without passing through the same point twice. While the travel salesman problem is conceptually very simple, and easier to understanding why solving it directly implicates trade in the real world, what may not be so obvious is the fact that this is an np-hard problem by definition, and therefore, has a hypothesis space too big to come up with a close solution. We can, however, optimize.

Optimum solution: Genetic Algorithm.

While all 4 algorithms were tested, the Genetic Algorithm was the one that performed the best. This may be because the Travelling Salesman problem has its domain on the discrete values, not continuous. Therefore, we shall expect a poor performance from algorithms such s MIMIC. Because we are not allowed to visit the same point twice, we can consider this a vector problem, where the direction in which we are heading matters. That may be why the Genetic Algorithm performs best here.





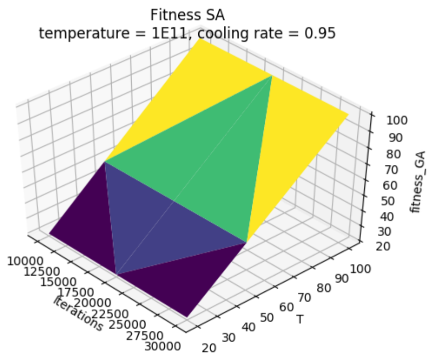
In the graphs above we see the algorithms form multiple different angles. Quit literally, we can observe is performance decaying almost exponentially when the number of points increase, and logically enough, improving performance if we allow enough iterations and time.

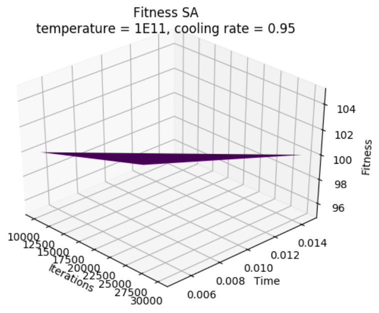
Problem: Continuous Peaks.

This is also a hard problem with many local optimum, which adds uncertainty for optimization algorithms since we want to find a balanced between finding the maxima, and stopping at an appropriate level. More specifically, the Continuous Peaks problem is at 1D space and imposes the question of which is the highest peak with a certain search space.

Optimum solution: Simulated Annealing.

While all the algorithms were able to converge to a maximum after a certain number of iterations, the Simulated Annealing did find que maxima of 100 the quickest. Tis may have to do with its cooling rate being set to 0.95, a ratter aggressive number. The algorithm therefore had more incentive to ignore local optima and find a global one.



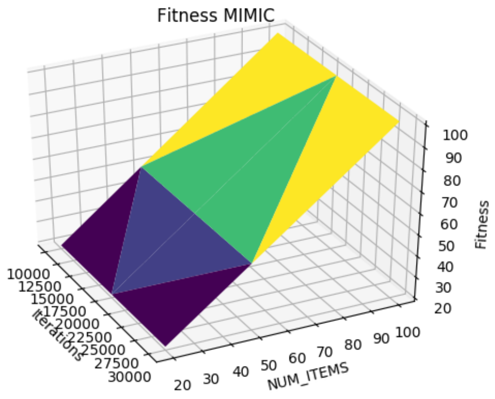


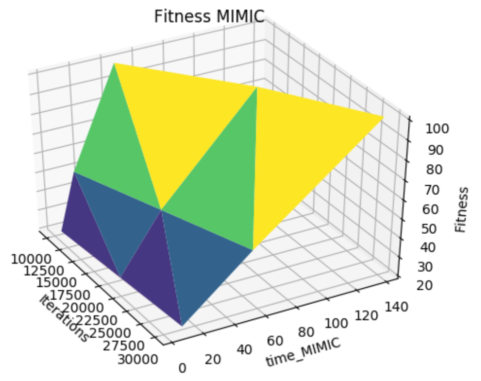
It is also interesting to notice the almost linear relationship the algorithm has with its fitness function, indicating that even before 1000 iteration it already had a high fitness, and that more iterations will most likely not yield in a new maximum.

Problem: Knapsack.

Knapsack is also an np-hard problem, and therefore has a really big hypothesis space. It relates well with the travelling salesman problem, expect that now we are in a scalar space, and therefore the order of operations do not matter.

Optimum solution: MIMIC.





The optimal algorithm here was MIMIC. Even though the search spaces is really big, given enough time and iterations, MIMIC was able to leverage its probability search functions to find the best solution among the algorithms.