Everardo Villasenor

CS 7641

Spring 2020

Assignment 2: Randomized Optimization

Dataset Analysis

The dataset I chose is the default of credit card clients found in the UCI Machine Learning Repository. This data set tries to predict whether an account will default (Yes = 1, No = 0) based on 23 features. The features include the amount of the given credit, gender, education, marital status, age, history of past payments delays, amount of previous payments and bills. There are 30,000 samples within this dataset with 78% negative samples and 22% positive samples. This is an interesting dataset because it represents a practical problem that is difficult for risk managers to forecast because of the vast number of features and potentially infinite unique situations. This problem is interesting because credit card default is a rare event and there are many different paths to get the outcome we want to predict. I did conduct some basic exploratory data analysis and found that some features are highly skewed and will probably need to transform the data in order to make use of some machine learning algorithms.

Methodology

Four local random search algorithms are implemented and then analyzed for certain optimization problems. The problems are generated using the mlrose-hiive library. Then three of them will be used to find good weights for an artificial neural network (ANN). Training, cross-validation score and run times will be used to track how effective the ANN is and then I will compare it to the backpropagation technique used in Assignment 1. Learning and validation curves are also generated to compare. I initially used only one hidden layer with TODO: 100 nodes for the backpropagation technique used in Assignment 1. I will used other variations of this parameter because I did not fully explore this in Assignment 1.

Optimization Problems

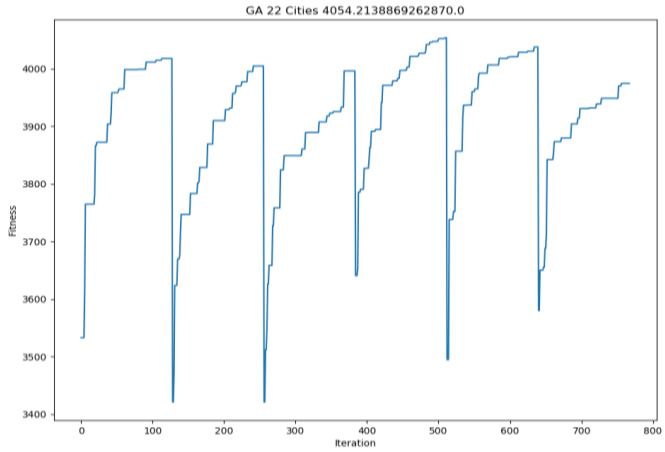
Optimization problems are ones where the objective is to find the best state of a fitness function which we are trying to maximize. The optimization problems I chose are the traveling salesman, knapsack and flipflop problem.

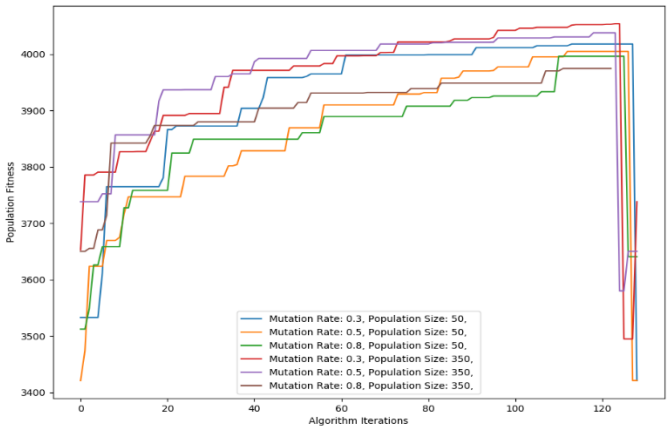
Traveling Salesman

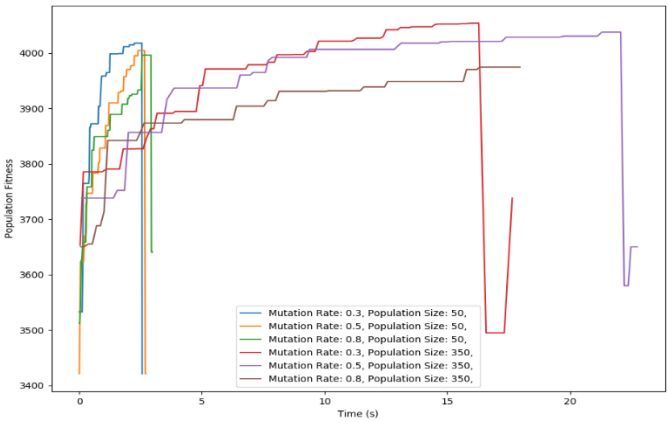
This is a classic problem where the goal is to visit every city in the problem space defined in the shortest amount traveled, although for the purposes of the grader the fitness function is optimized by the longest distance traveled visiting each city just once. This is a classic NP-hard problem and very aptly solved with the help of random optimization algorithms. I found MIMIC to find the best score (4054.21) in the best time.

Genetic Algorithm

This algorithm attempts to remember states that maximized the function and try to come up with consecutively better states therefore, I expect this optimization algorithm to be good at finding the maximum of this problem because it is important to remember the states. The best fitness this found was 4054.21 as the maximum distance and it found it in 17.64 seconds using a population of 350 and a mutation rate of 0.3. From my experimentation it seems that this problem prefers the parameters for the genetic algorithm to be sampling from a large population and moderate mutation rate. Too high of a mutation rate and the algorithm is too random and takes more time to converge, and too low of a mutation rate then the algorithm never fully explores the problem.

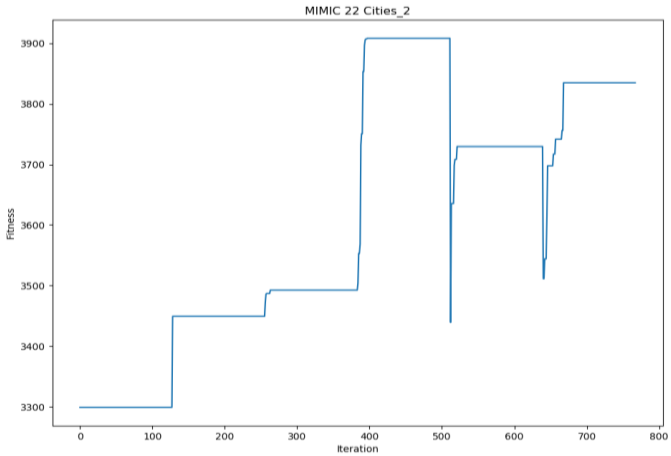


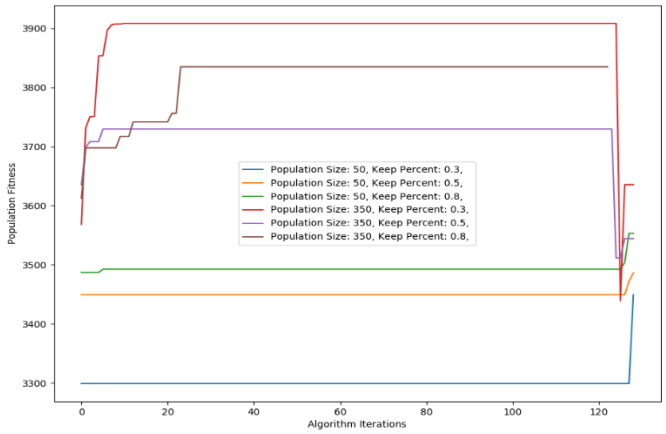


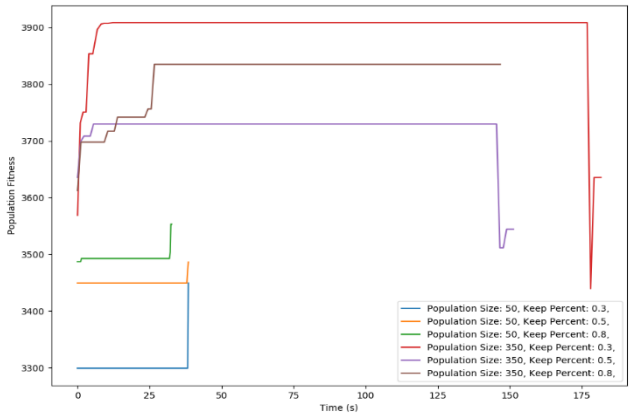


MIMIC Algorithm

The best fitness score for this algorithm is 4054.21 and it found it in 15.32 seconds. This algorithm tries to keep previous information and samples from regions of input space that contains the maximum for some fitness function. This sample represents the most likely best fit by using a probability distribution above some threshold. Initially, the probability distribution is uniform but with more estimates it can vary the shape of the distribution. I did use the same parameters as GA and used the fast version of the algorithm and the best was the population size of 350 and mutation rate of 0.3.

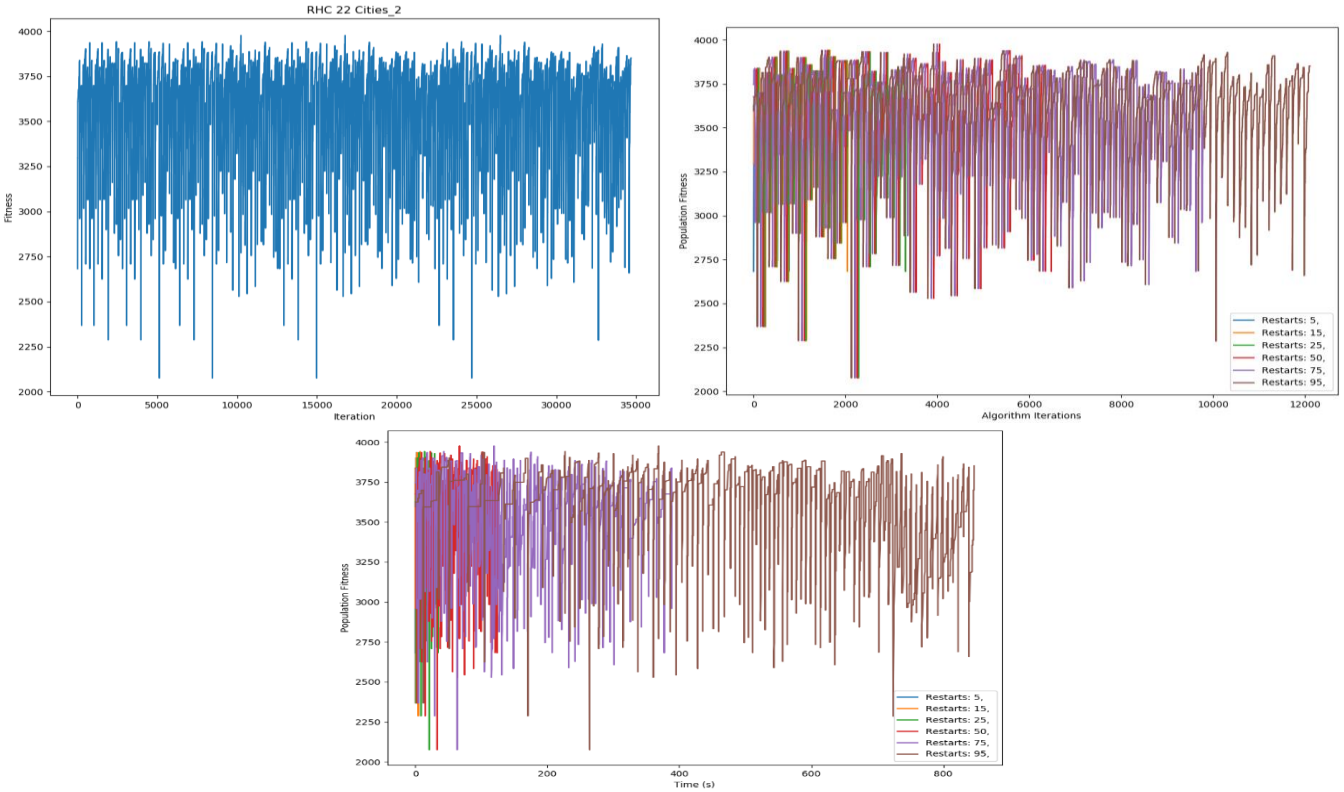






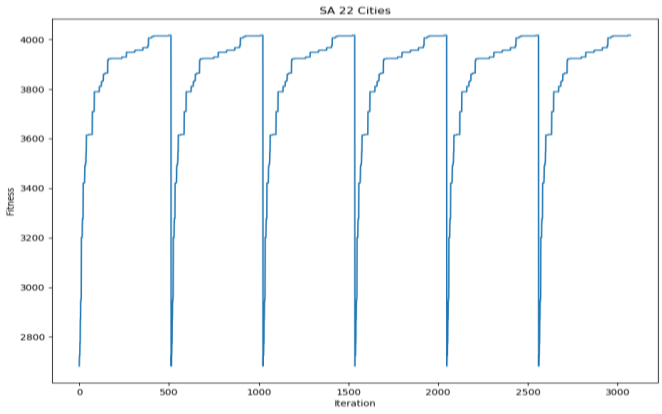
Randomized Hill Climbing Algorithm

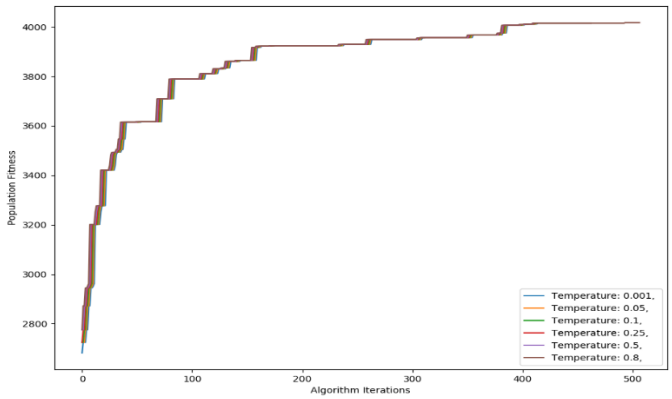
The best score this achieved was 3976.30 and in a time of 203.95 seconds with 95 restarts. This makes sense because it took a lot to overcome local optima by using more restarts because it is a greedy algorithm. In general, this algorithm takes longer to converge for problems where a very small subset of initial values leads to global optima and the majority lead to local optima.

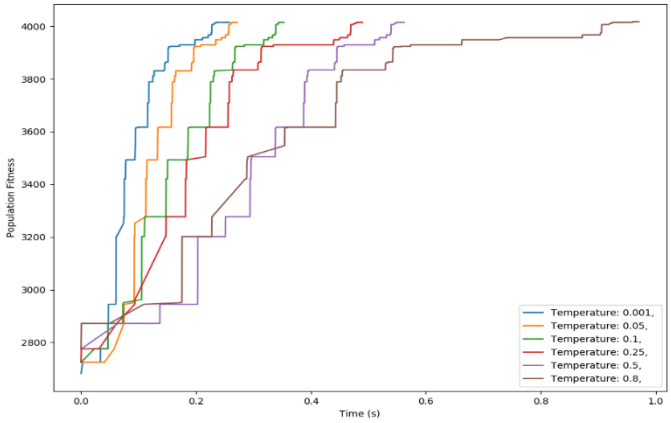


Simulated Annealing Algorithm

The best score is 4017.11 and it achieved it in a time of 0.85 seconds. Although this was not the highest score found for this problem, it is close and found it very fast. This algorithm does a good job of forcing to explore the problem space and it is tuned by the temperature and cooling rates chosen. I chose a geometric decay cooling which is fast for all my experiments with simulated annealing and a variety of different starting temperatures (0.001 – 0.8). At higher temperatures the algorithm can explore the hypothesis space more and the fast cooling rate means that the algorithm will no spend as many iterations deviating from optimal paths.





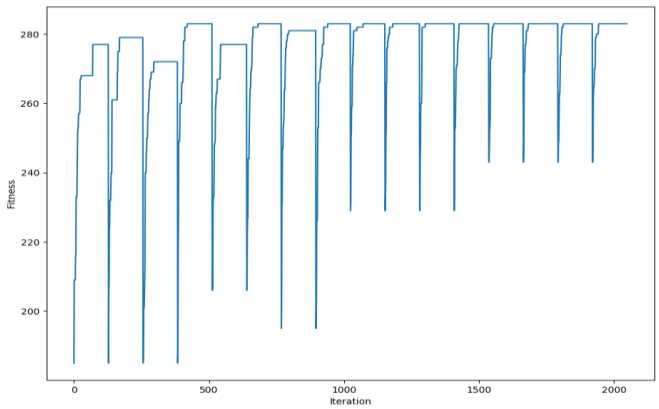


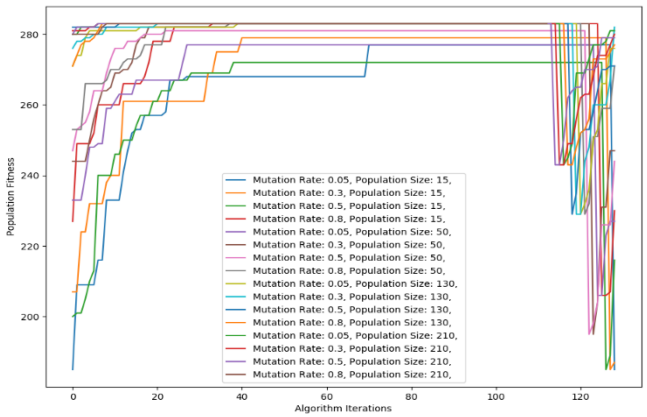
Knapsack

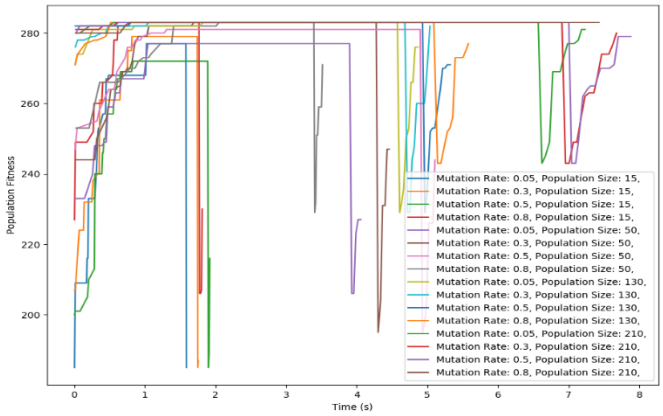
This problem attempts to maximize a fitness function defined by a set of items where each item has a known weight and value that is needed to fit into a knapsack with a maximum capacity. It is another NP-hard problem that is helped using random optimization algorithms. The constraints I chose was a max item of 6, max weight per item is 26, and max value per item is 12. The best algorithm I found in terms of fitness score and time is the genetic algorithm.

Genetic Algorithm

This algorithm should do well because it selects parents who did well at maximizing the fitness function and crosses them over with some mutation for randomness, which fits this knapsack problem because the problem constraint is counterproductive to the goal of maximizing value. Therefore, remembering states that did this well gives this algorithm an advantage. It got the best fitness of 283.0 in a time of 0.78 sec using a mutation of 0.8 and a population of 210. The results are found below for several parameters. What I noticed is that the higher the mutation rate the faster the algorithm can find the best fitness and the larger the population the better for selection of states to get the best fitness. This makes sense as a larger selection and high diversity with mutations will select a good state to optimize the problem.

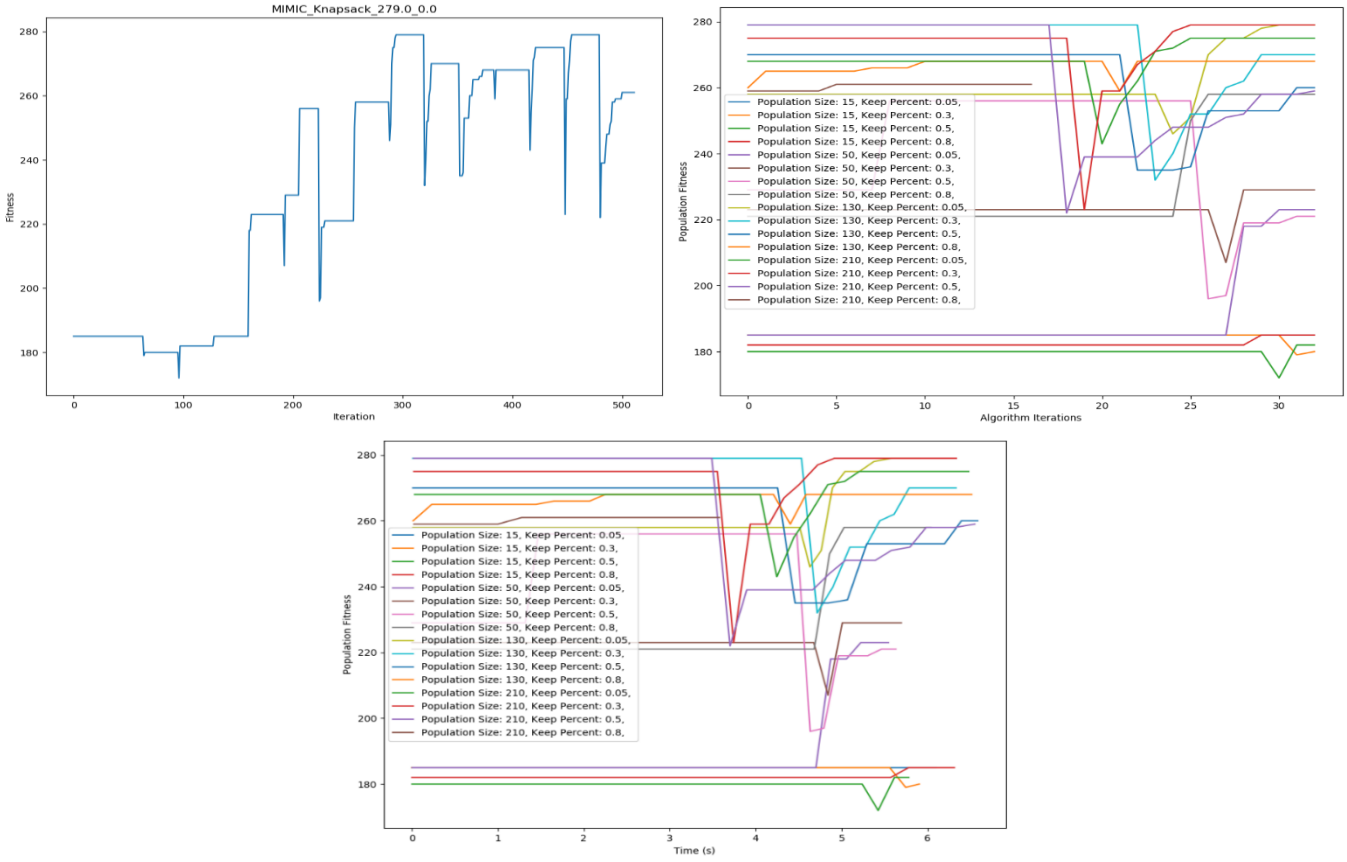






MIMIC Algorithm

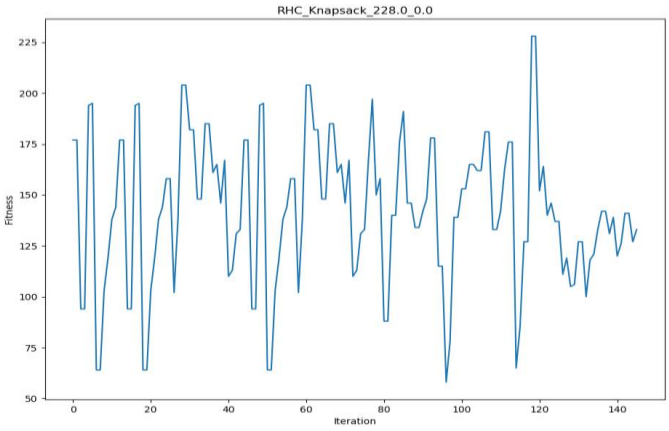
This algorithm keeps track of previous states by using a probability distribution instead of using parents to cross different states like in the genetic algorithm approach. Tracking the previous state is important in an NP-hard problem like Knapsack so I expected it to do well and I found that it got a fitness score of 279.0 in the best time of 1.22 seconds using population size of 210 and keep rate of 0.5. Generally, a moderate keep rate and large population offers the best results. This makes sense because a larger population to sample from is beneficial for this algorithm and it needs some randomness from the keep rate because otherwise a high weight and high value item might be kept in a knapsack when multiple lower value and lower weights items might increase the score. Too low of a keep rate and the algorithm does not retain useful information for this problem. This algorithm is like the genetic algorithm, but I was surprised that it did not do as well with respect to score or time complexity. My guess is that the genetic algorithm benefited greatly from the randomness in the algorithm. The results for my testing can be found in the graphs below.

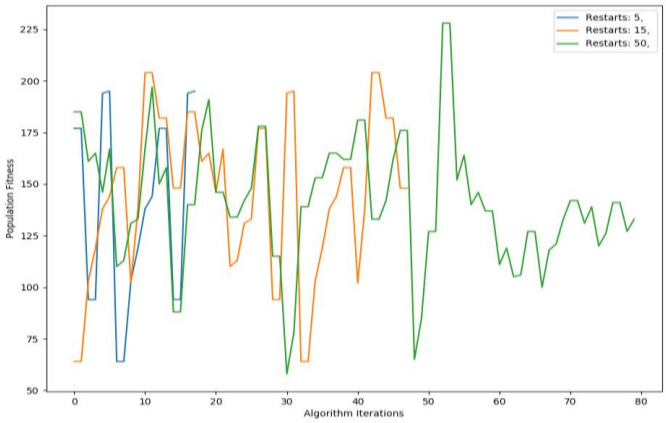


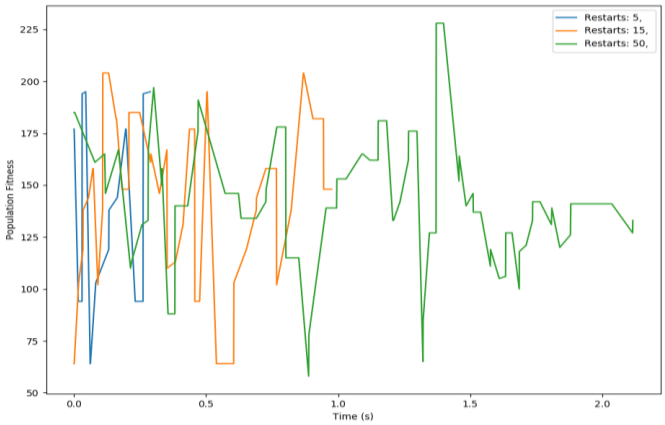
Randomized Hill Climbing Algorithm

This is a simple algorithm that starts at different points in order to find the maximum for the fitness function. I didn't think it would fair well for this problem because it could make a decision of filling the knapsack with a high value item

with a high weight and think that's the highest fitness it could achieve and not remember maybe smaller weighted and high value items to incorporate in the knapsack. The best fitness score it achieved was 228.0 in a time of 1.87 seconds. Therefore, the algorithm did not perform as well as GA or MIMIC in fitness score or time. From the graphs below on can see the high randomness of the algorithm and how the more restarts the more probable it is to get a higher score, but it will also take inherently more time.

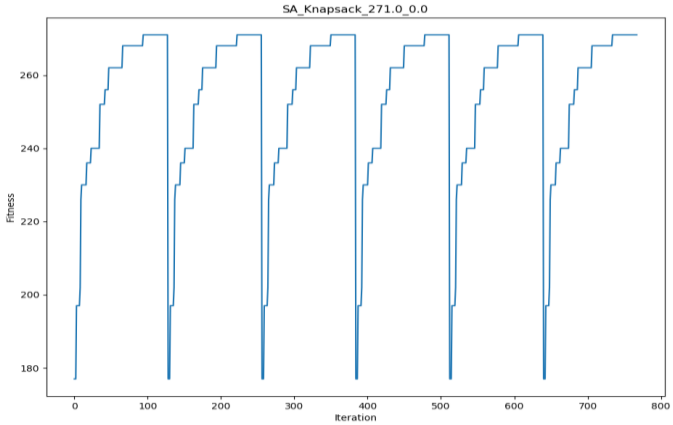


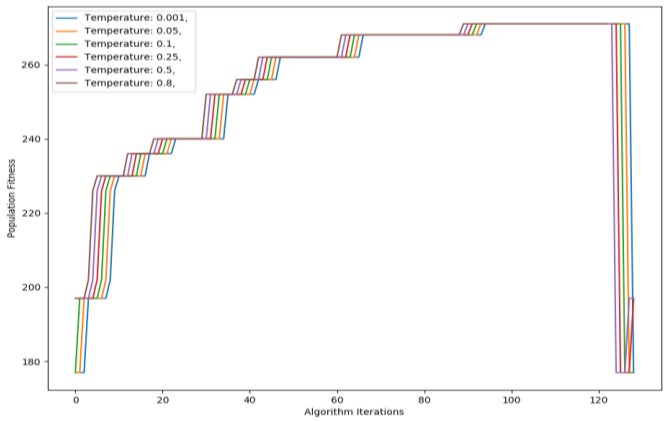


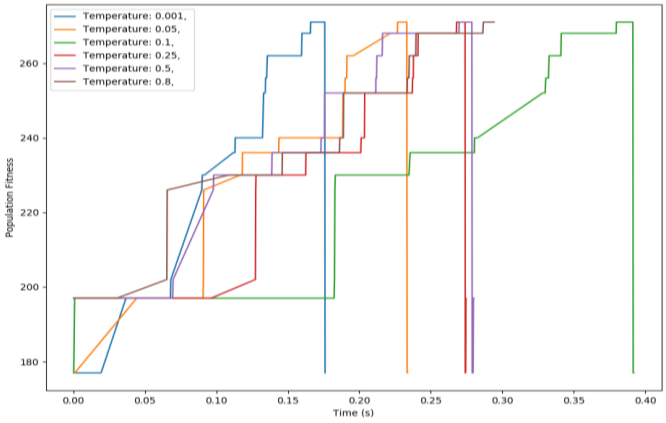


Simulated Annealing Algorithm

This algorithm is useful for finding global optima in the presence of a lot of local optima. It is like RHC, but it makes a calculated risk of moving in a bad direction in the hopes that it will improve. Instead of where RHC will only move in a good direction. It found the best fitness at 271.0 which is not too far from the maximum found by GA of 282.0 and using less time. It did so at low temperatures which means that it acted in a greedy fashion and less random if it started with a higher temperature. Since it is greedy and not dynamic, then it will likely not get the best solution of an NP-hard problem like this. The results of my experimentation are illustrated in the graphs below.





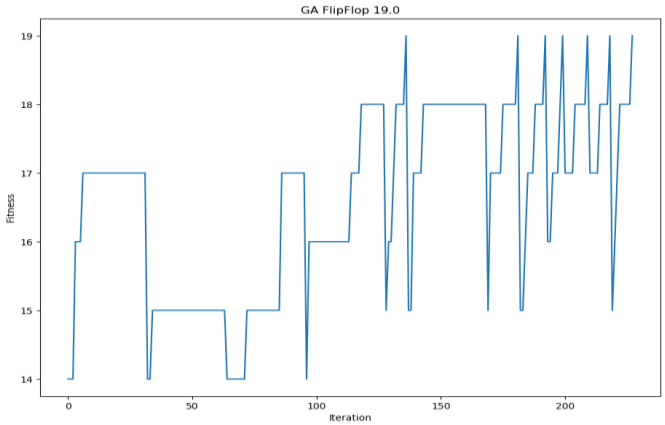


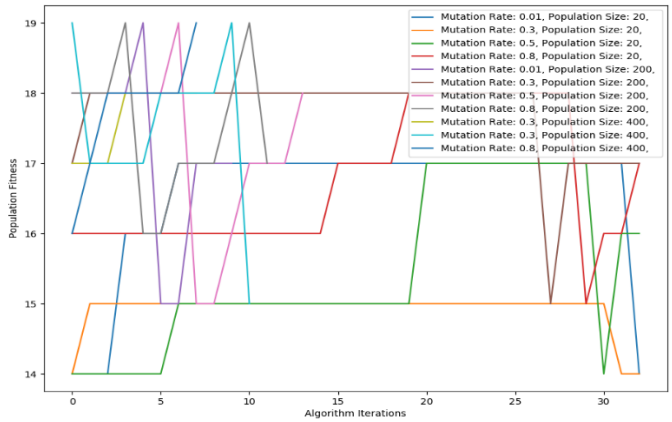
Flip Flop

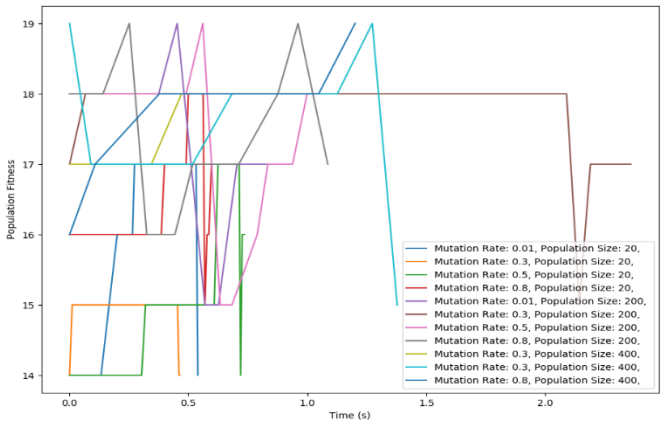
This is a simple bit problem where consecutive numbers in a series cannot be the same. I would expect that for many a problem space it would be important to remember states that work, but it is a simple constraint therefore a simple random algorithm might do the trick. I found that simulated annealing worked the best in terms of time and score. This makes sense because it does not need to keep track of previous states and can overcome many local optima to find the global optima. I found that simulated annealing worked very well for this problem in terms of fitness and time.

Genetic Algorithm

The best fitness score achieved was 19.0 and it achieved it in an average time of 0.807 seconds. This algorithm took time to find the best score as it first it explored worse solutions which makes sense because it tries to cross bits of parent states to get the best. For this problem it is critical where it chooses to do the crossover because the constraint is among consecutive bits.

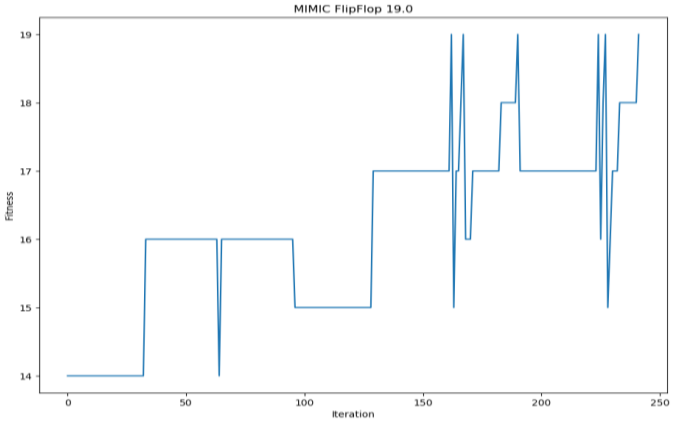


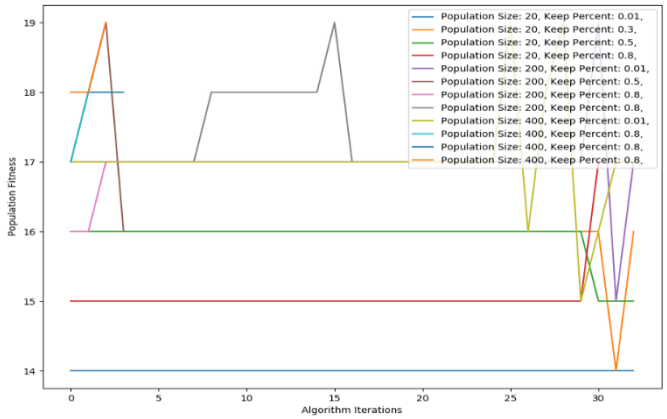


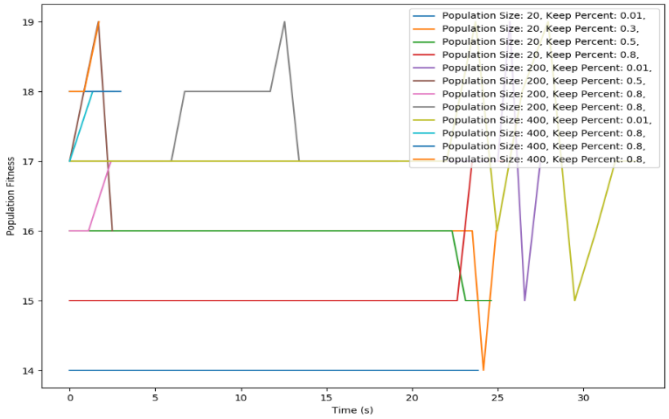


MIMIC Algorithm

The best fitness score achieved was 19.0 and with an average time of 4.55 seconds. This algorithm must keep track of distributions and it takes a long time to find the optimal score even though it finds it so another algorithm such as simulated annealing is better.

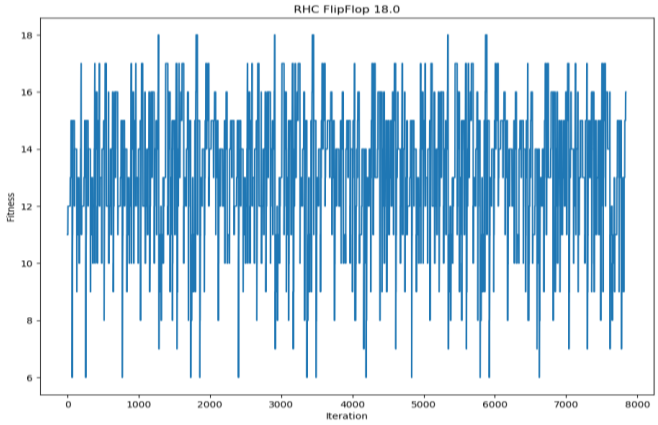


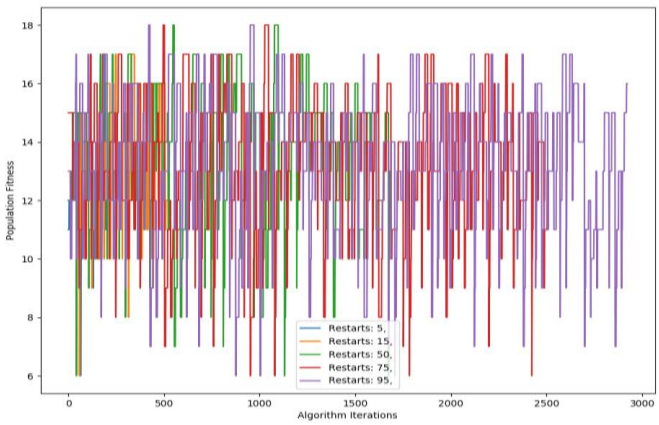


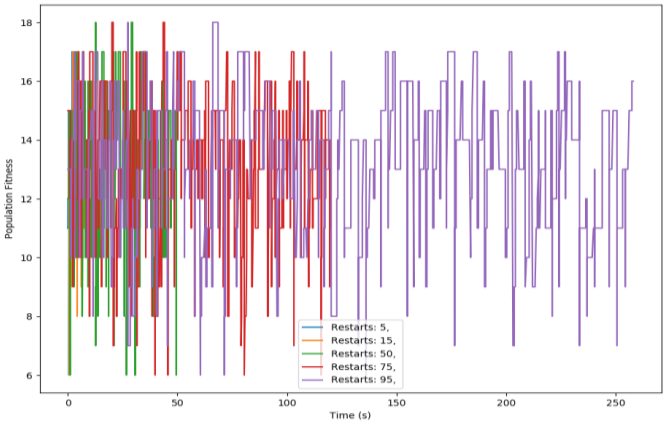


Randomized Hill Climbing Algorithm

The best score this algorithm achieved was 18.0 with a time of 32.2 seconds. It did not explore enough of the states and it already was taking a lot of time compared to simulated annealing.

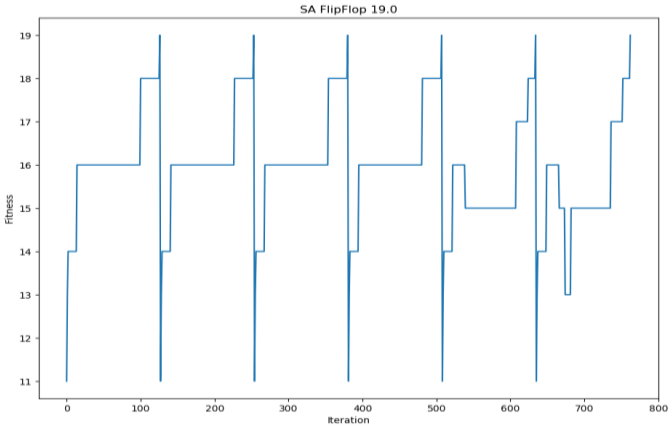


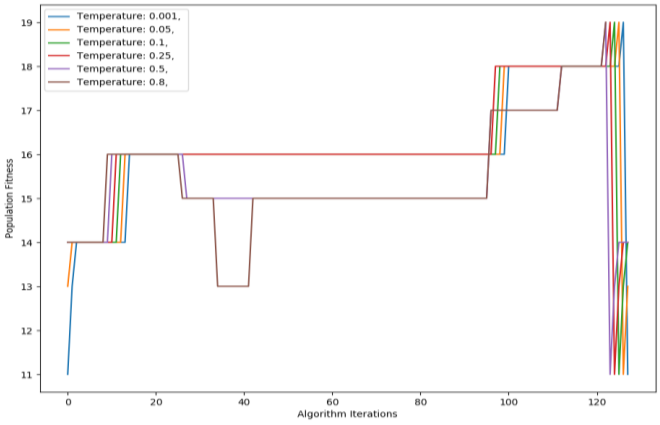


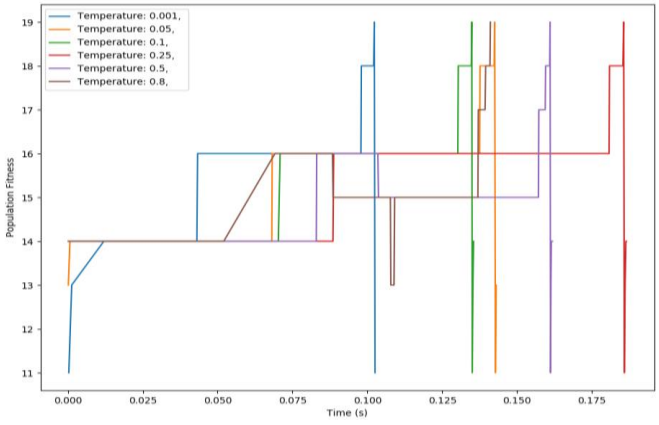


Simulated Annealing Algorithm

The best score achieved by this was 19.0 and a time of 0.48 seconds. It achieved the best score in the shortest amount of time. This is a stark contrast to its most similar algorithm in RHC and due to its ability to overlook bad local decisions in search for a good global one. One can see from the results below that even with a relatively low temperature it finds the best fitness, which means it does not have to explore much more than how RHC makes decisions to find the global optima.





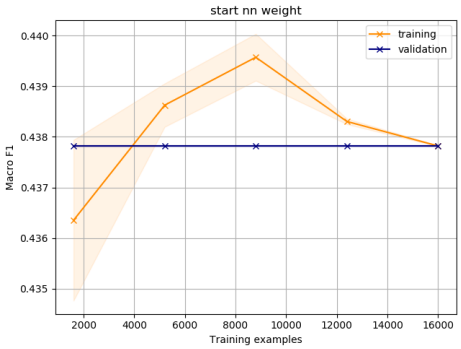


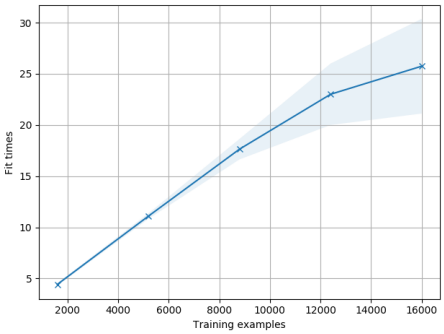
Local Random Search vs. Backpropagation

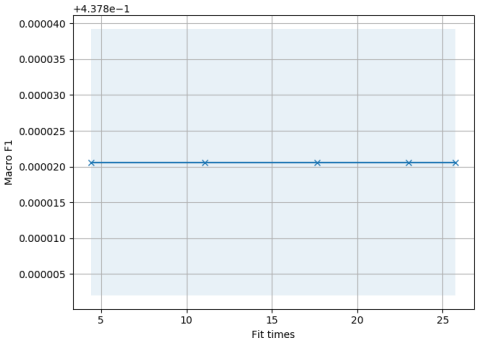
I explored using the three optimization algorithms (RHC, SA, GA) and compared it to the performance from the ANN used in Assignment 1. The backpropagation ANN had parameters of one hidden layer with 100 nodes and the relu activation function. This achieved a test score of 0.62 ± 0.02 and after having a train score of 0.62 ± 0.02 with a mean fit time of 55.23 ± 1.11 seconds. The GA algorithm to get the optimized weights scored 0.62 ± 0.02 and a mean fit time of 2052.44 ± 379.88 seconds. It was using a mutation probability of 0.25. The discrepancy in time without a benefit in performance means that this algorithm is not a good choice to use in application. The SA algorithm got a score of 0.61 ± 0.005 and a fit time of 74.4 ± 21.1 seconds. The parameter for the algorithm was using a geometric decay schedule with

a decay of 0.5. The RHC algorithm got a score of 0.61 ± 0.005 and a fit time of 107.79 ± 26.02 seconds. These two algorithms are similar, so it was not a surprise that they had similar performance and it was like the backpropagation and GA method therefore, it is not that interesting to analyze. The more interesting analysis is on the fit times which both were faster than the GA algorithm but not as fast as backpropagation. If I had to choose, I would use SA for optimizing the weights, but there is a reason why backpropagation is used in the real application.

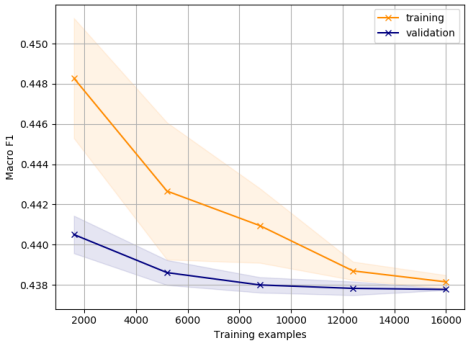
Backpropagation

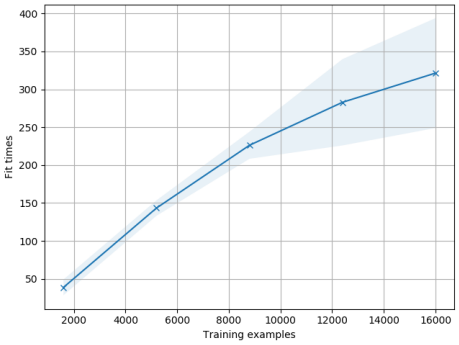


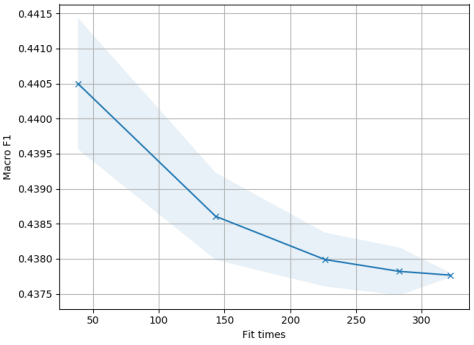




Synthetic Annealing







Random Hill Climbing