CS7641 Assignment 2: Randomized Optimization

Steel Ferguson
Sferguson42@gatech.edu

Introduction

In this analysis I investigate behaviors of four different Random Optimization (RO) algorithms: Random Hill Climbing (RHC), Synthetic Annealing (SA), Genetic Algorithm (GA), and Mutual Information Maximizing Input Cluster (MIMIC). I selected 3 simple bit string problems to highlight the strengths of SA, GA and MIMIC. I then apply RHC, SA, and GA to a different optimization problem: calculating optimized weights for the Neural Network (NN) from Assignment 1. With the NN problem, I also compare versus the Gradient Descent (GD) algorithm for weight selection. In each of the problems we measure success using a fitness function designed for the problem For the NN, the fitness is the classification accuracy.

The Four Peaks Problem and Synthetic Annealing

This first problem, the Four Peaks (4P) problem, highlights a strength of Synthetic Annealing. In the 4P problem, there are four ways to achieve a local maximum with two only two of those being global maxima. Specifically, given a bit string input and a number parameter T, the global maxima are found with T leading 1s followed by all zeros or T trailing ones preceded by all zeros. The local maxima are of all 1s or all 0s. Given this structure there is a large basin of attraction towards the local maxima. If an algorithm consistently chooses neighbors with a higher value, there is a strong chance that it will end up in a local maximum (that is not a global max).

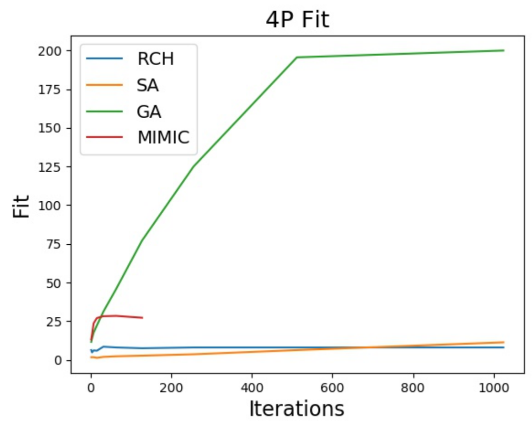
SA allows for choosing neighbors that are not the higher than the current position. Because of this it is more likely to not ‘get stuck’ in a basin of attraction of a local max. Given an appropriate starting temperature (T) and cooling/ decay rate (D), the algorithm will reach the global maximum more often than hill climbing. By comparing SA vs RHC on a 4P problem various time (10 here) at increasing maximum iterations we can compare average performance between the algorithms.

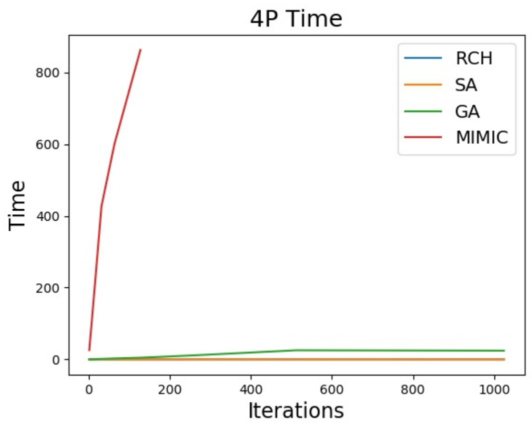
In figure ‘4P Fit’ below, we see some evidence of SA reaching a global max more frequently than the RHC algorithm. Although the fitness values are dwarfed by the more complex GA and MIMIC algorithms, the SA outperforms RHC in the long run. In the shorter run, we can RHC outperforms SA on average. We would expect that due to the “greedy” nature of RHC (always choosing the higher value in the short run). We would also expect SA to rise higher than RHC in the long run because it allows for lower fitness in the beginning (while T is high) with the hopes of landing in the basin of attraction that will lead to the optimal solution.

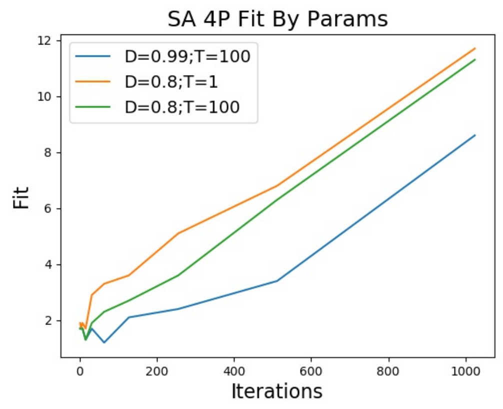
In the Figure ‘4P Time’ we see the average time needed to achieve given max iterations. When using a scale that includes time needed for MIMIC, the difference in time needed between SA and RHC is trivial. In practical terms they are very similar. We could expect a longer time to converge at a maximum (especially if T is high and/or D is low) because SA allows for taking a detour to lower areas in order to achieve the global maximum more frequently.

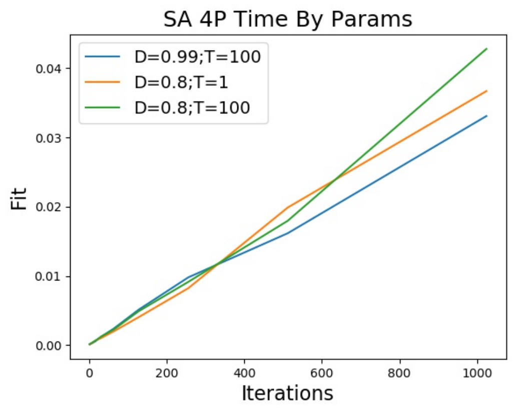
For using the SA, it is important to tune the T and D parameters for the problem at hand. In this problem, T needed to be higher than the default setting of 1 in mlrose-hiive with a lower D. This allows for more switching and going lower early on (high T) but for that stage not last quite as long (lower D). A comparison of fitness by iterations for 3 permutations in Figure 'SA 4P Fit By Params' illustrates this point. Figure 'SA 4P Time By Params' also shows the time requirement difference with these different settings. We can expect some more calculations and movements in the earlier stages with higher T (more available options to move). That is illustrated by the green and blue being slightly higher in the first ~300 iterations.

I will also note that as we increase the size of the problem (200 bits in this example) to 1000 bits, we see an increase in the time required for all algorithms. That required by SA does not stand out in particular.









Knapsack Problem and the Genetic Algorithm

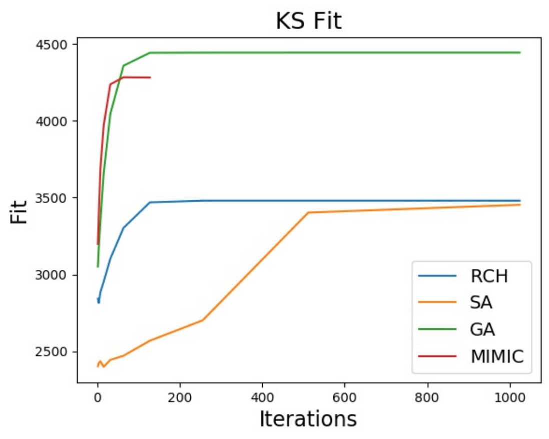
Although the Genetic Algorithm performs well on a variety of problems, the Knapsack problem (KS) illuminates the strength of the GA well. The KS is created by giving a set of weights and values that can be thought of as items to put in a knapsack. The fitness is given by the total sum of values of items (here we limited to 1 of each item) unless the weight of the items is over a given amount. If the weight is too much the total fitness is zero. The problem is very complex with large “drop offs” if you were to think of the fitness as a 2D curve.

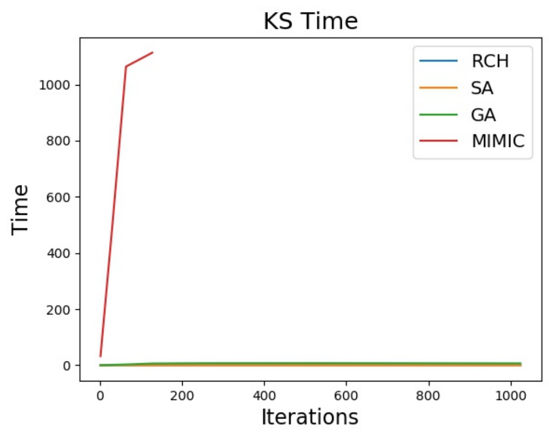
The GA is adept to handle a complicated problem such as this. There are many items that could go in the knapsack, so greedy algorithms will likely put items into the sack that were not the best items. The GA uses samples to find areas with strength. It then allows for “cross over” and “mutation” to (ideally) get the best of both samples. Due to this feature, the GA will more commonly pick up on value items to put into the Sack that may be “far away” from a neighbors perspective. This gives GA an advantage on the KS problem.

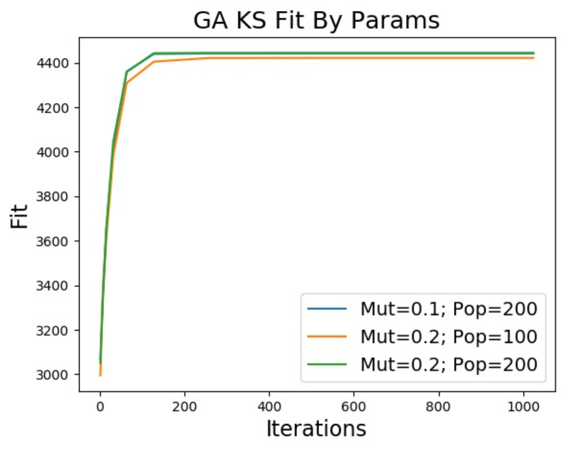
In Figure ‘KS Fit’ below, we observe that GA achieves the highest fit value. Fitness achieved by GA is decidedly higher than RHC and SA which seem to get stuck on local maximuma. The combining of samples allows GA to not get stuck as they do. The MIMIC algorithm does achieve higher in the early iterations, but we cannot say MIMIC achieves fitness “faster”. In Figure ‘KS Time’ we see again that the time required by MIMIC dwarfs the time required by other algorithms. For this problem, not only does GA achieve better fitness after ~50 iterations but it also achieves higher fitness orders of magnitude faster than MIMIC. GA has a more complex structure that does require analyzing a population of neighbors (possible steps), but the process is significantly faster than that of MIMIC.

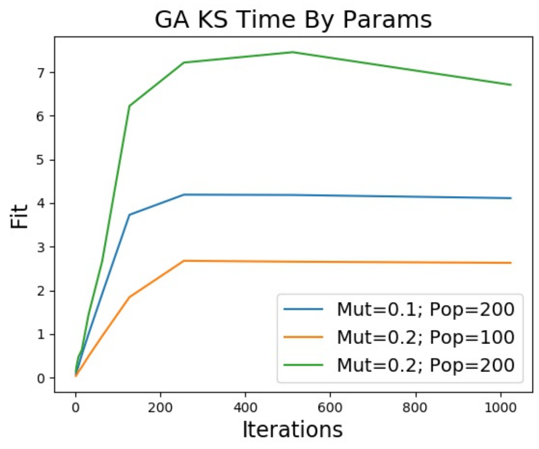
In tuning the GA, key parameters for this (and other) problems include the size of the population sampled (Pop) and the rate of mutation (Mut) allowed. For this particular problem Pop=200 and Mut=0.2 yielded the best results. Figure ‘GA KS Fit By Params’ shows how better fitness is achieved in fewer iterations and is maintained than by lower values of Pop and Mut. We can also observe in Figure ‘GA KS Time By Params’ that grabbing a large population and performing more mutations requires more time from the algorithm (as the green line is higher than the blue or yellow lines). That is expected as it is performing a more complex task at each iteration.

Increasing the size of the problem does have an impact to time required by the algorithms, and this certainly holds true for the GA.









The Flip Flop Problem and MIMIC

The MIMIC is a brilliantly designed algorithm that takes into account a couple of blind spots of other algorithms. Other algorithms will iterate multiple times and land on a new answer, but they do not keep track of what they have learned. They also do not consider the structure of the data and neighbors near maxima. The MIMIC retains information from neighbors sampled and creates a structure to describe the distribution in the fittest parts of the sample.

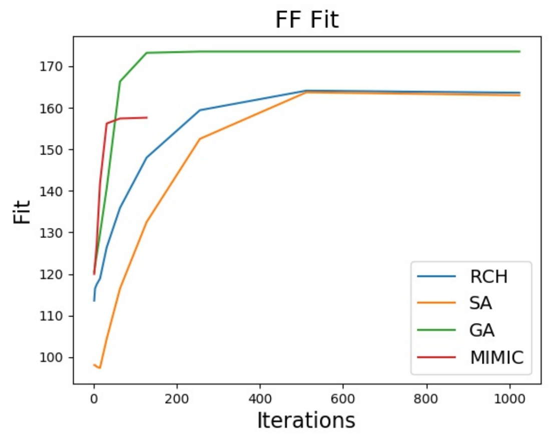
The Flip Flop (FF) problem gives a global maximum with the maximum number of changes from 1 to 0 in bit string. The FF problem highlights the MIMICs ability to detect structure and distributions. At each step the MIMIC will take a sample, keep the values with the highest fitness values, and then find a distribution for the structure. The FF global maximum is very structured in that it entirely depends on the bit string on either side, and it requires a clear pattern to achieve.

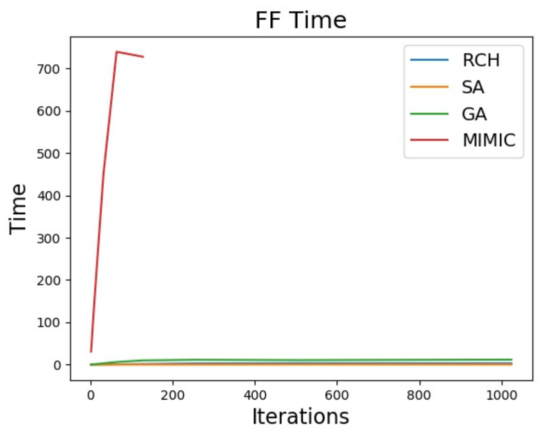
In Figure ‘FF Fit’ we can see a high fitness in the first iterations of the FF. The GA algorithm surpasses the score of the MIMIC around ~50 iterations similarly to the KS problem, and RHC and SA also eventually achieve a fitness higher than the MIMIC achieves in the first 128 iterations. These results were contrary to my hopes but also yield important insights. Specifically the advantage of fitting a distribution on the highest scoring neighbors in a sample would likely be better showcased in a problem where the structure is not the same across the entire distribution. We could think of it as a landscape that has features with higher fitness. The MIMIC would be able to isolate the features and create a structure to estimate the distribution in those areas. In the FF, the structure does not have such features.

Similar to result in other problems, the MIMIC requires far greater time per iteration than the other algorithms considered, as we can observe in Figure ‘FF Time’. This makes sense as the MIMIC requires fitting a distribution at each iteration (after sampling and filtering). In order for the MIMIC to yield a higher fitness in less time, the fitness needs to be many times greater than the alternatives. If time is a prime concern, MIMIC may not be the best solutions. However, in cases where evaluation of fitness is costly (e.g. requires human input or costly testing), MIMIC may be the best solution. Specifically, it could yield a higher fitness in very limited iterations in this problem or yield a higher fitness overall in others.

Important parameters in tuning the MIMIC are the population size taken and kept. In this problem I found optimal parameters as sampling 100 and keeping half. Larger populations of samples require more time and fitness evaluations. Larger percent to keep also requires more time as it increases the population size that it fits a distribution to.

Increasing the size of the FF problem requires more time from all algorithms, as expected. In the case of the MIMIC, we do see a significant increase in the amount of time required. MIMIC took hours on my computer for simple cases. The MIMIC took the majority of a day to finish FF in a larger form.





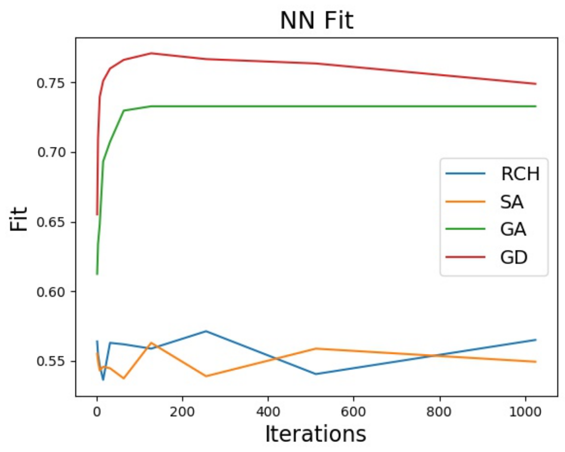
Using Random Optimization to Set Weights of a Neural Network

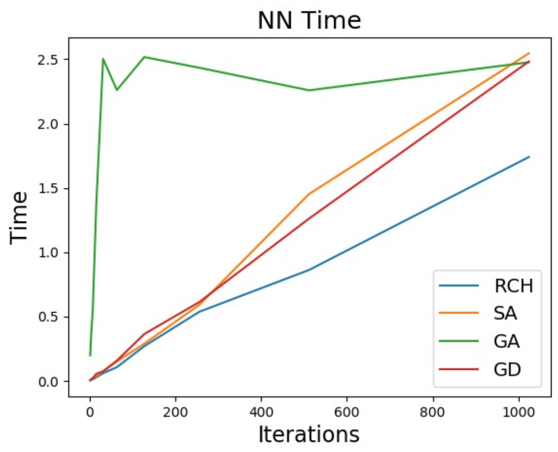
In the first assignment I used a data set from Pima Native Americans with medical information and the classification of having Diabetes or not. In this assignment I have used the same data set to find weights for a Neural Network classification model. As I used mlrose—hiive on this assignment instead of scikit-learn, I fined tuned a new model with the defaulted gradient descent algorithm to find the structure. I then tuned each model for each algorithm used (except for the hidden nodes structure which I kept constant at one hidden layer of 9 nodes. As the outcome was fairly balanced ~60-40, I used accuracy as a measure of fitness (instead of F1 score or another alternative).

As observed in Figure 'NN Fit', the defaulted Gradient Descent (GD) outperformed all of the algorithms explored in this analysis. GA came the closest. As each point in the chart represents an average of trials, we would more confidently go with the results, but I would point out that in some of the trials GA did achieve fitness higher than the GD on average. GD makes changes to the weights in the 'direction' that yields the most change, and we would expect that to works in this case in the vast majority of the cases. I searched for a different structure of the network that might yield a higher accuracy on average than the GD and I was unable to find it for this particular dataset. That does not preclude the possibility of a GA algorithm outperforming GD in the task of assigning weights to a NN.

Each algorithm more or less converged to a fitness in the first 50-80 iterations as we can see in Figure 'NN Fit' with the steep increases followed by relatively flat fitness. This makes sense as there were only 9 nodes in 1 hidden layer. We could expect more iterations needed for more complicated network structures.

As shown in Figure 'NN Time' we see similar time required per max iteration for all algorithms considered except GA. We see GA taking a long time for even a low number iteration (~30). As it converges, we do not see a significant increase in time. We see a notch at 64 iteration that we can attribute to randomness in the random optimizer as there is not a structural reason for less time for more iterations.





Conclusion

In the NN, I was unable to beat the gradient descent algorithm’s performance on my dataset, but I did find the GA particularly strong I a variety of problems.