Assignment 2 : Randomized Optimization (CS7641)

Stella Lai-Hoong Soh

lsoh3@gatech.edu

Abstract

This report seeks to analyze the behavior of 4 randomized optimization algorithms - Randomized Hill Climbing, Simulated Annealing, Genetic Algorithm and MIMIC - tested on 3 optimization problems - **Continuous Peaks**, **Flip Flop** and **Knapsack -** and neural network with weights optimization.

1. Optimization Algorithms

The optimization algorithms and optimization problems are implemented using the Python library of mlrose_hiive.

1.1 Randomized Hill Climbing (RHC)

As explained in the video lecture, randomized hill climbing involves iteratively moving towards more optimal neighbors until a peak is reached. Each time RHC starts with a random initial guess and moves towards the direction of increased fitness. This optimization problem uses very little memory.

1.2 Simulated Annealing (SA)

Simulated Annealing approximates the global optimum of a given function. The annealing procedure refers to the real life Physical Annealing whereby an optimal arrangement of metal particles (when the potential energy of the mass is minimized or the minimum lattice energy state) is reached as metals cool down after being subjected to extremely high temperatures. The SA problem iteratively compares the output of the objective function running with current and neighboring point, and if the neighboring point yields a better output than the current one, that neighboring point becomes the base for the next iteration. In this assignment, we could easily tune the parameters such as the annealing schedule (mapping of time to temperature and how fast the temperature decreases).

1.3 Genetic Algorithm (GA)

Genetic Algorithm is a search heuristic reflecting Charles Darwin's theory of evolution, in that the fittest individuals are selected for reproduction in the process of natural selection. The algorithm starts with an initial population, randomly explores, and allows mutation and

crossover (just like in the world of biology) on parent chromosomes based on their fitness scores.

1.4 Mutual-Information-Maximizing Input Clustering (MIMIC)

MIMIC communicates information about cost function in an "efficient and principled" way. In each iteration, the algorithm generates a density estimator, which generates additional samples, and in turn establishes a new threshold which then constructs a new density estimator. Such a density estimator, capable of representing clusters of highly related parameters, imitates the behavior of crossover. This is immensely powerful as clusters are learned from data, and not pre-defined by programming. Equipped with such sophistication to build structure and find optima, surely running this algorithm would take a long time.

2. Dataset Description

The wine quality dataset is a 1600 (rows) x 12 (columns) dataset that has been used in Assignment 1. It consists of input feature attributes such as volatile acidity, residual sugar, and pH etc. based on physicochemical tests. The last target column - the quality is assigned a 'good' or 'bad' string. The additional processing I undertook was assigning a 1 to the 'good' quality wines, and a 0 to the 'bad' quality wines.

2.1 Implementation of Neural Networks using Gradient Descent

After preprocessing the wine_quality dataset, I split the dataset into 80% training and 20% testing, just like I had done in Assignment 1. With mlrose_hiive's NeuralNetwork's class, weights have already been built-in into the hidden layers. I defined a hidden layer of 10 nodes, learning rate of 0.001 and used gradient descent for the algorithm field. The 3 randomized optimization algorithms - Random Hill Climbing(RHC), Simulated Annealing(SA) and Genetic Algorithm(GA) - were run with this neural network apart from gradient descent.

Randomized Hill Climbing

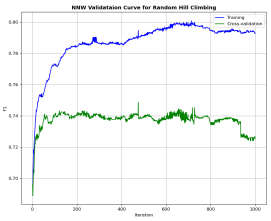

Fig. 1- Validation curve for RHC

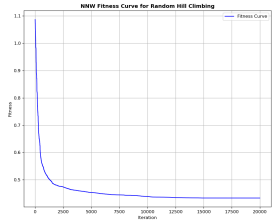

Fig. 2 - Fitness curve for RHC

As shown in Fig. 1 above, the training F1-score is much higher than the F1-score for cross-validation. However, (refer to Fig. 2) as the number of iterations increases, the fitness decreases, and converges to around 0.45.

Simulated Annealing

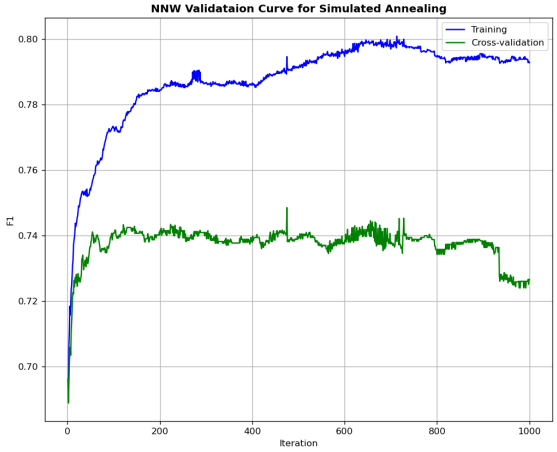

Fig. 3 - Validation curve for SA

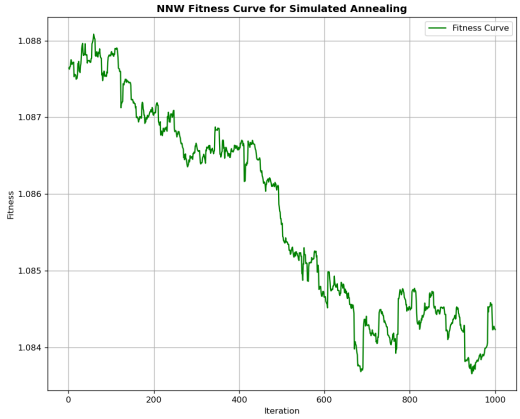

Fig 4 - Fitness curve for SA

As shown in Fig. 3 above, the F1-score for training is, again, much higher than the F1-score for cross-validation. I was expecting the fitness curve for SA to be much like Fig.2's fitness curve. For some reason, Fig. 4 shows otherwise. It still exhibits a downward trend, but I can't explain the "squggliness" of the graph.

Genetic Algorithm

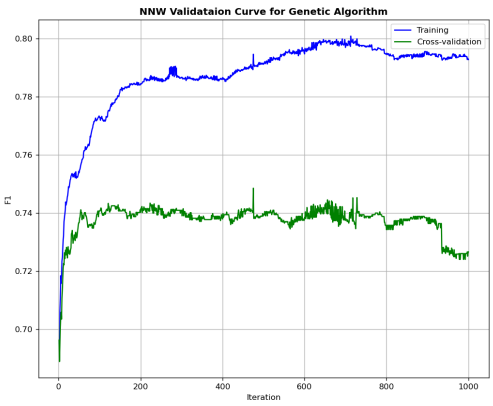

Fig. 5 - Validation curve for GA

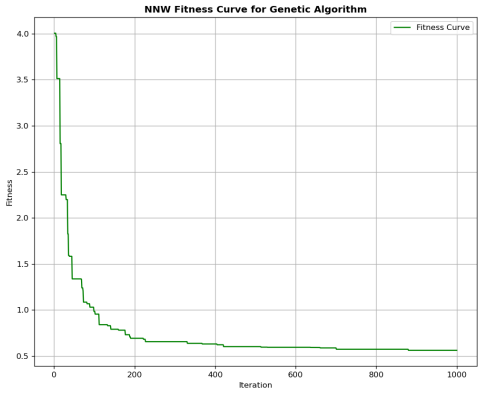

Fig 6 - Fitness curve for GA

Both validation and fitness curves for GA resemble those of RHC. Fig. 6's fitness curve for GA shows that as the number of iterations get closer to 1000, fitness score converges to around 0.6

Gradient Descent

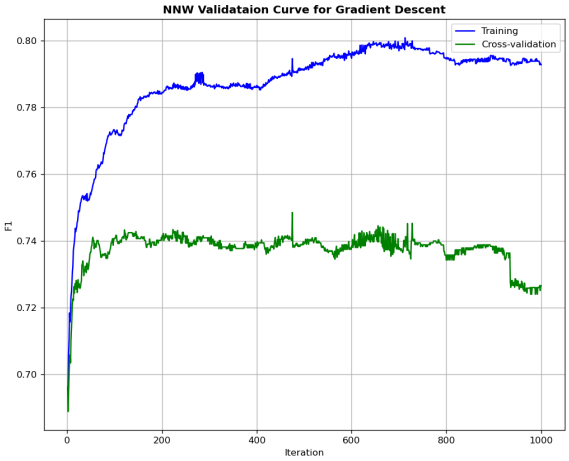

Fig. 7 - Validation curve for Gradient Descent

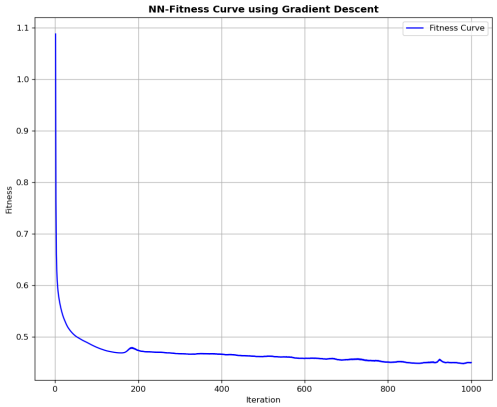

Fig 8 - Fitness curve for Gradient Descent

Validation curve and fitness curve for gradient descent resemble those of RHC. Fig. 8's fitness curve for gradient descent shows that as the number of iterations gets closer to 1000, the fitness score converges to around 0.45.

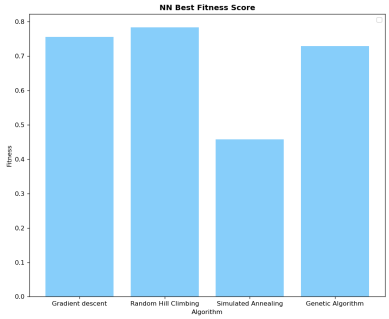

Fig.9 - F1-scores for all 4 algorithms

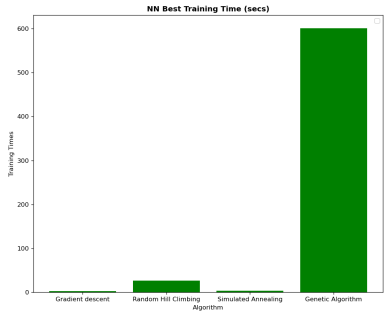

Fig. 10 - Running times for all 4 algorithms

From my exploration of sklearn's classification_report() in Assignment 1, I have come to appreciate the use of f-1 score. Prof. Isbell's statement of "the loss or only the accuracy won't tell you the whole story" prompted me to compare the algorithms' performance by looking at F-1 scores and running times. F-1 scores combine the precision and recall of a classifier or model into a single metric by taking the harmonic mean:

$$\text{F1-score} = \left(\frac{\text{Recall}^{-1} + \text{Precision}^{-1}}{2}\right)^{-1} = 2 * \frac{(\text{Precision} * \text{Recall})}{(\text{Precision} + \text{Recall})}$$

In most real-life cases, there are imbalanced class distribution, and it is crucial to know the measure of incorrectly classified items or the False Negatives and False Positives. [1]

I used F1-score to measure the quality of the trained neural network. Bar charts of the fitness and run-times of the 4 algorithms are as shown in Figs. 9 and 10 above.

The F1-score for RHC - 0.80 is the best, followed by gradient descent at 0.75, and GA at around 0.72. SA has the worst F1-score at 0.45. In terms of running time, GA took the longest time to run (600 secs.), with RHC being the second longest at around 30s. The other two algorithms - gradient descent and SA - took the shortest time to run.

For gradient descent, I have used a max_iterations of 1000, with learning_rate of 0.001, and max_attempts=100. For RHC, I have used a max_iterations of 20000, with learning_rate of 0.1, and max_attempts=1000. For SA, max_iterations of 1000, learning_rate of 0.001, and max_attempts of 500 were used. For GA, max_iterations of 1000, learning_rate of 0.001 and mutation probability of 0.1 were used.

In conclusion, the three things I found interesting are: (1) RHC, at a F1-score of 0.80, shows itself to be the winner, as opposed to gradient descent (2) For SA, with a learning_rate of 0.001, the "squiggly" fitness curve in Fig. 4, though exhibiting a downward trend, does not seem to converge. Given more time to explore, I would tweak the learning_rate some more to see if things change. (3) From Genevieve Hayes' article, the loss function takes the role of the fitness function. This confirms that the loss is kept to a minimum as we see the fitness curves of Figs. 2, 6 and 8 showing RHC, GA and gradient descent converging to some value.

3. Implementation of the 4 Algorithms in 3 Optimization Problems

The 3 optimization problems I have selected are: Continuous Peaks, Flip Flop and Knapsack. They are solved using the 4 algorithms: RHC, SA, GA and MIMIC. The best performing algorithm for each optimization problem is chosen based on its fitness value and running time.

3.1 Continuous Peaks Problem (highlighting GA)

A 4-peaks problem consists of 2 local optima with wide basins of attraction**.** The fitness of 4-peaks consists of counting the number of 0's at the start, and the number of 1's at the end , and returning the maximum. If both the number of 0's and the number of 1's are above some threshold value T, then the fitness function gets a bonus of 100 added to it.[2] *There are 2 small peaks where there are lots of 0's or lots of 1's, and 2 larger peaks, where the bonus is included.*
Continuous peaks problem is an extension of the 4 peaks problem, but contains many local optima. In a high-dimensional world, due to the presence of many local optima, solutions could be found in all dimensions. Continuous peaks problem highlights pretty well the differences between randomized optimization algorithms

I tested the 4 randomized optimization algorithms in a continuous peaks problem environment provided by mlrose_hiive's ContinuousPeaks(). Each run was for 500 iterations. For RHC, the best parameters were max_attempts=500 and max_iterations=500. For SA, the best parameters were init_temp=1.0*10**10, exp_const=0.99, min_temp=1.0 for exponential decay and arithmetic

[1] Refer to *Accuracy vs F1-score*

[2] Refer to *Evolutionary Learning*

decay, and init_temp=1.0*10**5, decay=0.99, min_temp=1.0 for geometric decay. For GA, the best parameters were mutation probability in the range of 0.1 and 0.2.

The results in Fig. 11 and Fig. 12 illustrate that **GA achieved the best fitness score** while RHC gave the worst. MIMIC performed the second best.

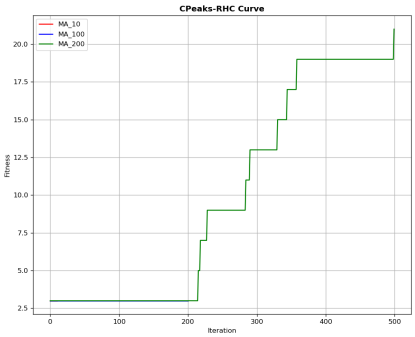

Fig. 11 - Cpeaks-RHC Fitness vs Iterations

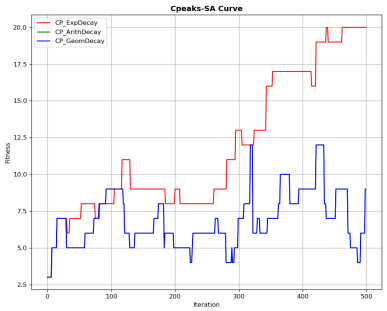

Fig. 12 - Cpeaks-SA Fitness vs Iterations

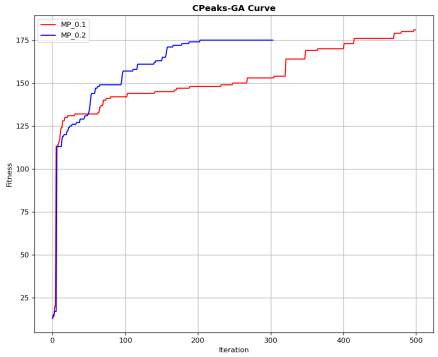

Fig. 13 - Cpekas- GA Fitness vs Iterations

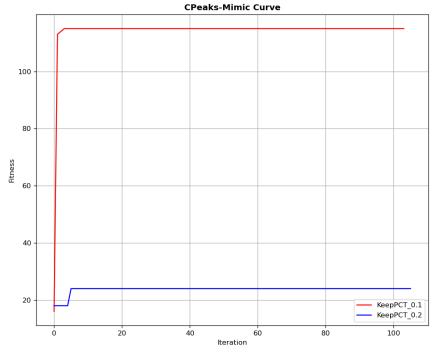

Fig. 14 - Cpeaks-MIMIC Fitness vs Iterations

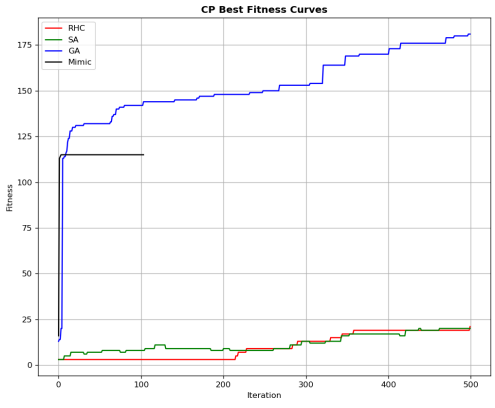

Fig. 15 - Best Fitness vs Iterations

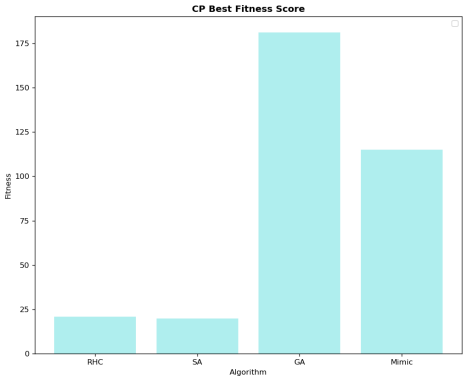

Fig. 16 - Fitness scores for 4 algorithms

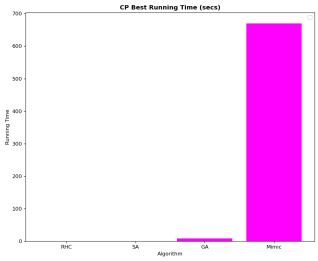

Fig. 17 - Continuous Peak's running time for 4 algorithms

I plotted the running time of all the 4 algorithms in a continuous peaks' environment. Fig. 17 shows that MIMIC was the worst while all the other 3 took a relatively short time to run.

3.2 Flip Flop Problem (highlighting MIMIC)

The flip flop problem is one that counts the number of times the bits alternates in a bit string. For example in a bit string, if a number flips to any other number number, this is counted as 1. A bit string that consists of alternating digits is one that has maximum fitness.

Using a problem size of 100, maximum iterations of 100, keep_pct range of 0.3 to 0.6, and a population size of 600, I tested all the 4 algorithms. I plotted the best fitness scores for all algorithms - RHC, SA, GA and Mimic in Fig. 18 below.

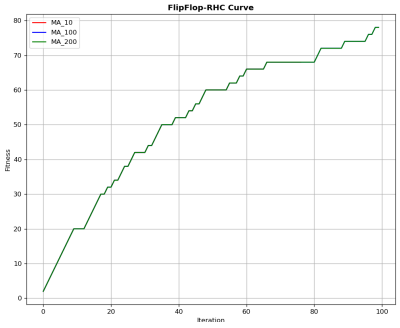

Fig.18 - Flip Flop-RHC Fitness vs Iterations

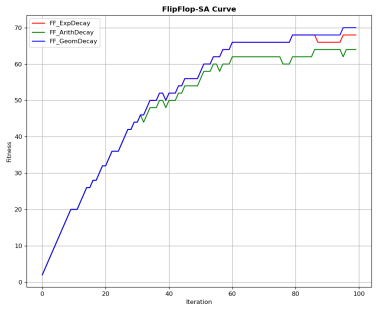

Fig. 19 - Flip Flop-SA Fitness vs Iterations

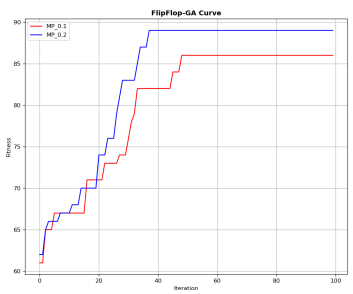

Fig. 20 - Flip Flop-GA Fitness vs Iterations

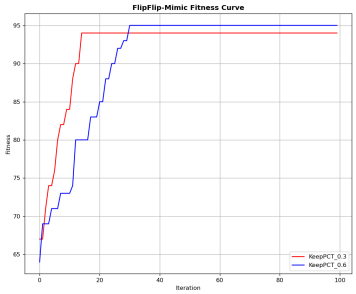

Fig. 21 - Flip Flop- MIMIC Fitness vs Iterations

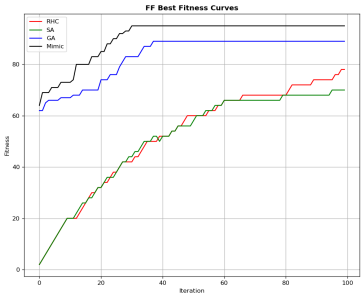

Fig. 22 - Best fitness vs. Iterations for 4 algorithms

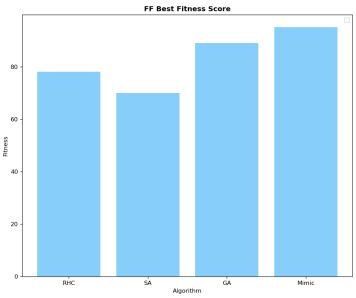

Fig. 23 - Flip Flop's fitness scores for 4 algorithms

The result shows that **MIMIC achieved the best fitness score** for flip flop problem, followed by GA , while SA performed the worst. Fig. 22 shows the best fitness score vs. iterations. As the number of iterations increases, we see MIMIC leading the pack converging at fitness score of 90, followed by GA converging at 85, RHC at 78 and SA at around 70.

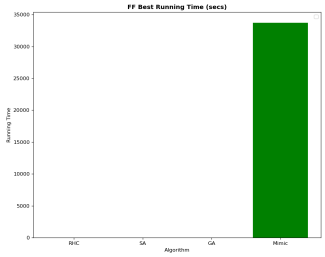

Fig. 24 - Best running time for all 4 algorithms

It is important to note that the clock time is considerably higher for MIMIC. As explained in Prof. Isbell's mimic-tutorial pdf[3], there are 2 components of MIMIC: "first a randomized optimization algorithm that samples from the input region space most likely to contain the optima for C(); second an effective density approximator that can be used to capture a wide variety of structure on the input space, yet is computable from second order statistics of the data."

Very likely, due to the fact that MIMIC constructs density approximators for each iteration, it comes as no surprise that MIMC takes a mighty long time to run. Looking at Fig. 20 above, MIMIC took a ~34000 seconds, which is approximately 9.44 hours to run while the other 3 algorithms took much less to run in the same environment.

Similar to the continuous peaks problem above, MIMIC algorithm took the longest time to run. However, in this case, MIMIC has the best fitness score out of all 4 algorithms. One thing the reader should note is: Parameter selection ultimately determines what the outcome would be. With a different set of parameters (i.e. a different keep_pct range or a different population size, for example), running in this flip flop problem environment, MIMIC may not be the algorithm that shows the best fitness score.

[3] *Professor Isbell's mimic-tutorial.pdf*

3.3 Knapsack Problem (highlighting SA)

The knapsack problem is an NP-hard problem[4] that has a polynomial time approximation. I find it easiest to think of the analogy of a thief on a robbing mission. There are finite items available to be robbed. Each item has its own weight and value. The thief has the task of filling the knapsack with a determined capacity such that he does not exceed the maximum weight the knapsack is capable of holding. The thief's profit is then calculated by summing up the value of all the items that he robbed. In short, items placed in the thief's knapsack must maximize total value without exceeding the weight limit of the knapsack. Further applications in business include maximizing profits for a catering service (for example) that wants to consider optimizing the amount of ingredients for a banquet while minimizing the amount of waste or unused ingredients that go into the compost bins.

I used mlrose_hiive's Knapsack() fitness function. The problem size was set to 1000, and the maximum iterations for each run was set to 5000. For RHC, (refer to Fig. 25), the best parameter is when max_attempts = 200. For SA, (refer to Fig. 26) the best parameters were for geometric decay, whereby the initial temperature was at 1.0*10**11 and decay rate of 0.99. For GA, (refer to Fig. 27), the best parameters were for mutation probability of 0.001 and population size of 25.

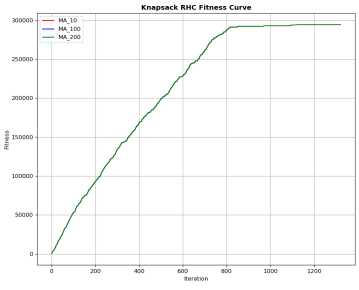

Fig. 25 - Knapsack- RHC fitness vs iterations

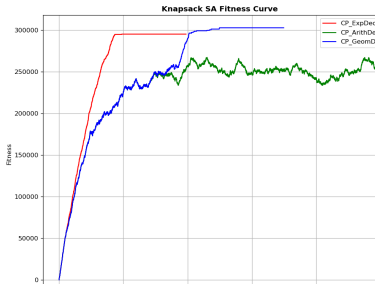

Fig. 26 - Knapsack - SA fitness vs iterations

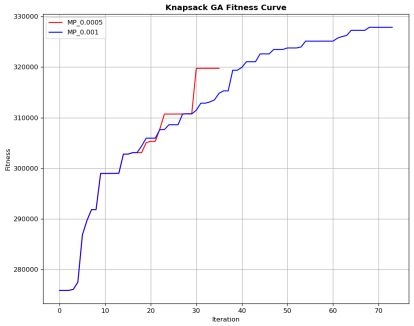

Fig. 27 - Knapsack - GA fitness vs iterations

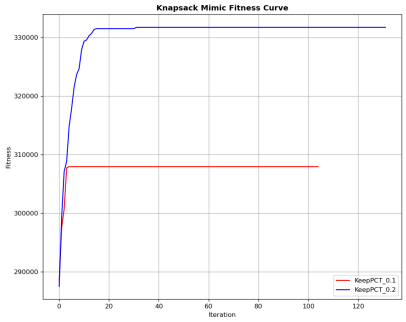

Fig. 28- Knapsack - MIMIC fitness vs iterations

[4] *Knapsack Problem*

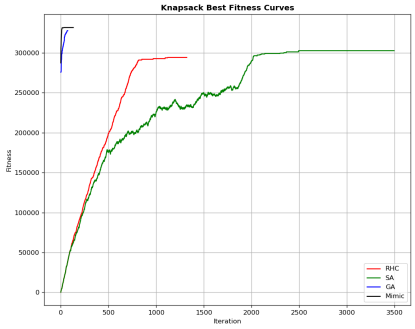

Fig. 29 - Best fitness vs Iterations for 4 algorithms

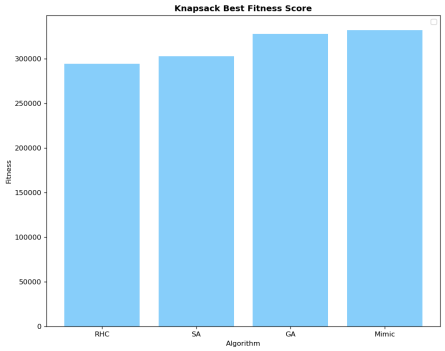

Fig. 30 - Knapsack's fitness score for 4 algorithms

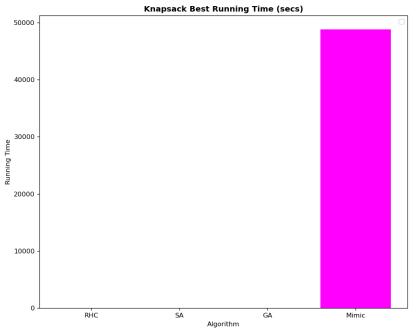

Fig. 31 - Best running time for all 4 algorithms

I was hoping to use the knapsack problem to highlight Simulated Annealing. At the very first run (the baseline), the problem_size was set to 100, max_iters=100. For SA, the initial temperature was set at 1*10**5, with exp_const=0.005 and decay rate=0.99. The baseline graph of fitness score for all 4 algorithms is as shown in Fig. 32:

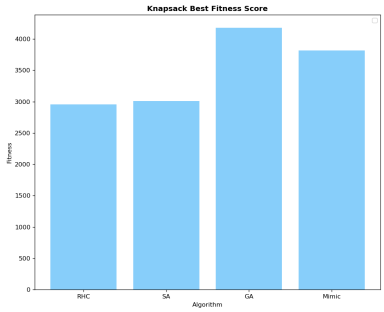

Fig. 32 - Knapsack's best fitness score for 4 algorithms (baseline)

After 12th attempts of tweaking over 2 days, and running knapsack test continuously for 24.5 hours, the best fitness score I could get is as shown in Fig. 30. The performance of SA and GA depends highly on parameter selection. Although in Fig. 25, SA still shows up to have 30000 for fitness score, and is not in fact the leader, I believe that if I were to tweak the parameters some more, and given more time to run the experiment, SA's fitness score would improve much more beyond 30000.