Comparison of Randomized Optimization Algorithms

Rui Hu
rhu63@gatech.edu
CS 7641 Machine Learning Assignment 2

1 Introduction

This report presents an analysis on the performance of 4 selected random optimization algorithms. They are tested on three discrete optimization problems, namely: Four Peaks Problem, Flip Flop Problem, and Knapsack Problem. The random optimization algorithms in scope are: Random Hill Climbing, Simulated Annealing, Genetic Algorithm and Mutual-Information-Maximizing Input Clustering (MIMIC). Finally, we used the first three algorithms to find good weights for a neural network, and compared to those obtained from traditional gradient descent and back propagation.

2 The Randomized Optimization Algorithms

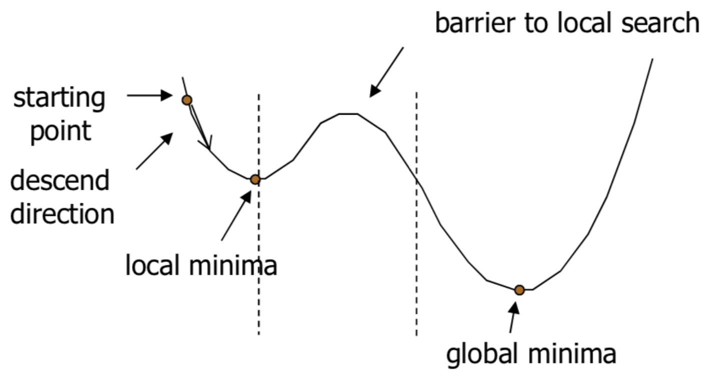
In this assignment, we measured the performance of 4 different algorithms on 3 different optimization problems and evaluated the performance of each algorithm in different situations.

2.1 Random-restart Hill Climbing (RHC)

Traditional hill climbing selects an initial point and then starts moving in the direction with the maximum fitness-score increase. However, hill climbing will not necessarily find the global maximum, but may instead converge on a local maximum, because it highly relies on the initial point. One way to fix this problem is to restart multiple times at different positions. Random-restart hill climbing iteratively does hill-climbing, each time with a random initial condition X_0. The best X_m is kept: if a new run of hill climbing produces a better X_m than the stored state, it replaces the stored state. Therefore, even if the algorithm doesn't discover the global optimum, it'll at least gets to a decent local optimum [1].

2.2 Simulated Annealing (SA)

Simulated Annealing mimics the Physical Annealing process and is used for optimizing parameters in a model. This process is very useful for situations where there are a lot of local minima such that algorithms like Gradient Descent would be stuck at.



The difference between hill climbing and SA is that SA provides extra energy to the searcher (in the form of temperature) that allows it to move to worse states. With this little trick the searcher can avoid getting stuck in a local minimum by jumping over small hills to see if there is a better optimal state. The temperature parameter gradually decreases to make sure the algorithm stops at some point. High temperature promotes exploration, low temperature promotes exploitation [2].

2.3 Genetic Algorithm (GA)

Genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection. Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as selection, crossover and mutation. The process begins with a set of individuals which is called a Population. Each individual is a solution to the problem you want to solve. In the selection phase, the fittest individuals are selected let them pass their genes to the next generation. In the crossover phase, for each pair of parents to be mated, a crossover point is chosen at random from within the genes. In the mutation phase, for certain new offspring formed, some of their genes can be subjected to a mutation with a low random probability [3].

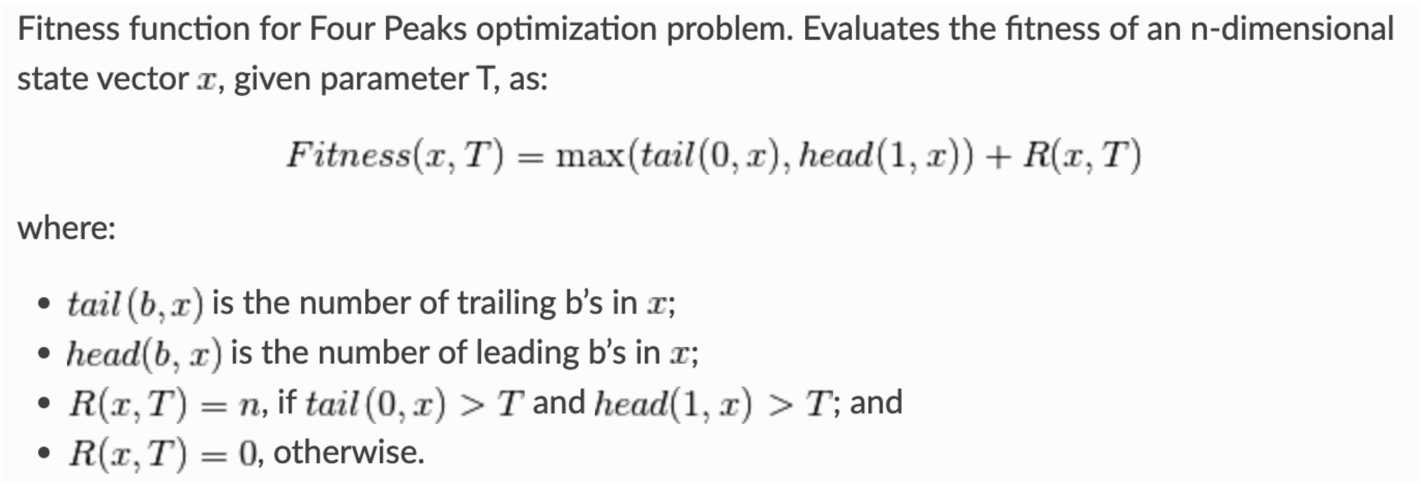
2.4 Mutual-Information-Maximizing Input Clustering (MIMIC)

Compared to MIMIC, one disadvantage of the abovementioned algorithms is that they learn nothing about the space they're exploring and don't keep track of the underlying probability distribution they're searching. The primary idea behind MIMIC is that it should be possible to directly model a probability distribution, improve and refine it over time, and end up with something that will keep the knowledge of the past iterations. MIMIC first randomly samples from those regions of the input space most likely to contain the optima for C(), and then used an effective density estimator that can be used to capture a wide variety of structure on the input space, yet is computable from simple second order statistics of the data [4].

3 The Optimization Problems and Experiment Results

In order to evaluate and test the performance of the previous algorithms 3 prob- lems are designed to show the strengths and weaknesses of each one of them.

3.1 Problem 1: Four Peaks Problem (FPP)



Fitness function for Four Peaks optimization problem. Evaluates the fitness of an n-dimensional state vector x, given parameter T, as:

$$Fitness(x, T) = \max(tail(0, x), head(1, x)) + R(x, T)$$

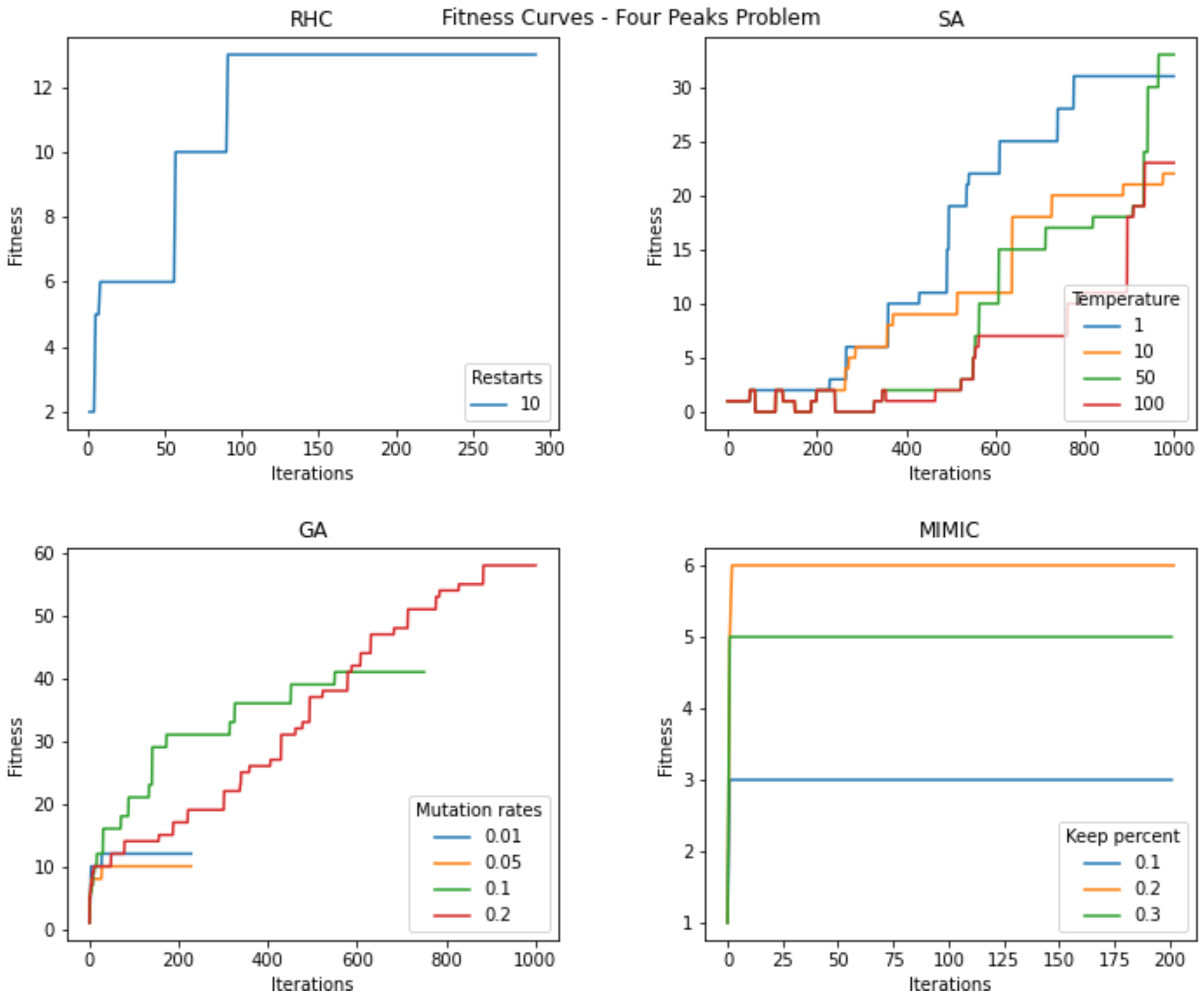
where:

- $tail(b, x)$ is the number of trailing b's in x;
- $head(b, x)$ is the number of leading b's in x;
- $R(x, T) = n$, if $tail(0, x) > T$ and $head(1, x) > T$; and
- $R(x, T) = 0$, otherwise.

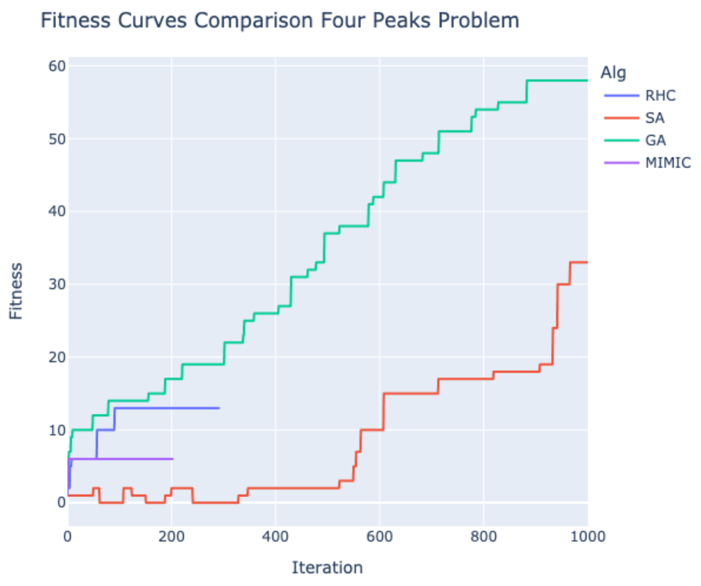
It's a problem with two local optima with wide basins of attraction designed to catch simulated annealing and random hill climbing. Genetic Algorithms are more likely to find these global optima than other methods.
It consists of counting the number of tailing 0s and leading 1s and returning the maximum, if both the number of 0s and the number of 1s are above some threshold value T then the fitness function gets a bonus of R=n added to it.

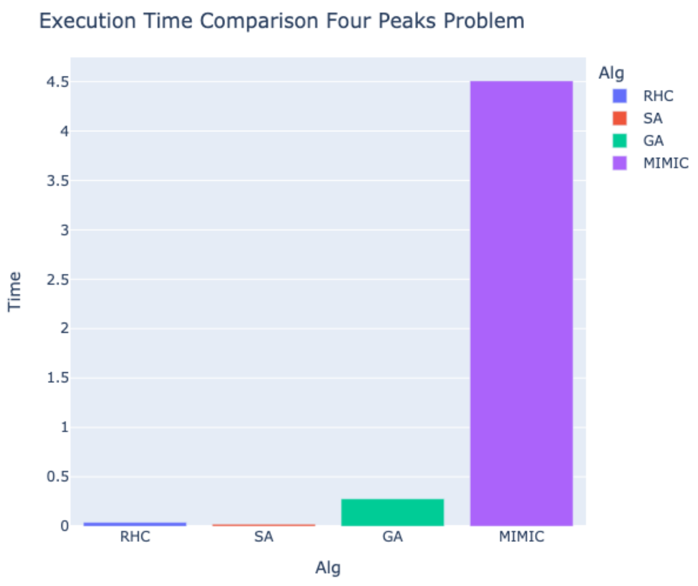
3.1.1 Results

We selected a Four Peaks Problem with length 100. This problem is more complex than flip flop problem. We tested the four algorithms with different parameters shown in the figure below. Simulated annealing has temperature from 1 to 100, genetic algorithm has mutation rate from 0.01 to 0.2, and the keep percent of MIMIC ranges from 0.1 to 0.3. For SA, the best parameter was Temperature=50. For GA, the best parameter was Mutation rate=0.2. For MIMIC, the best parameter was Keep percent=0.2.
The performance of SA and GA largely depends on their parameter selection. SA and GA performed relatively well with respect to fitness scores. RHC and MIMIC didn't achieve high scores. When search region is complex, a high mutation rate (0.2) can help GA overcome local optimum.



GA outperforms other algorithms in this experiment, with the highest fitness score, a moderate number of function calls and not much execution time.
Looking at the number of function evaluations completed by each algorithm, MIMIC and RHC have fewer iteration than GA and SA, but GA and SA achieved much better scores. Besides, MIMIC, took much longer time to run than the other 3 algorithms, since it's not only searching for optimal values but also recording the underlying distribution of the data.



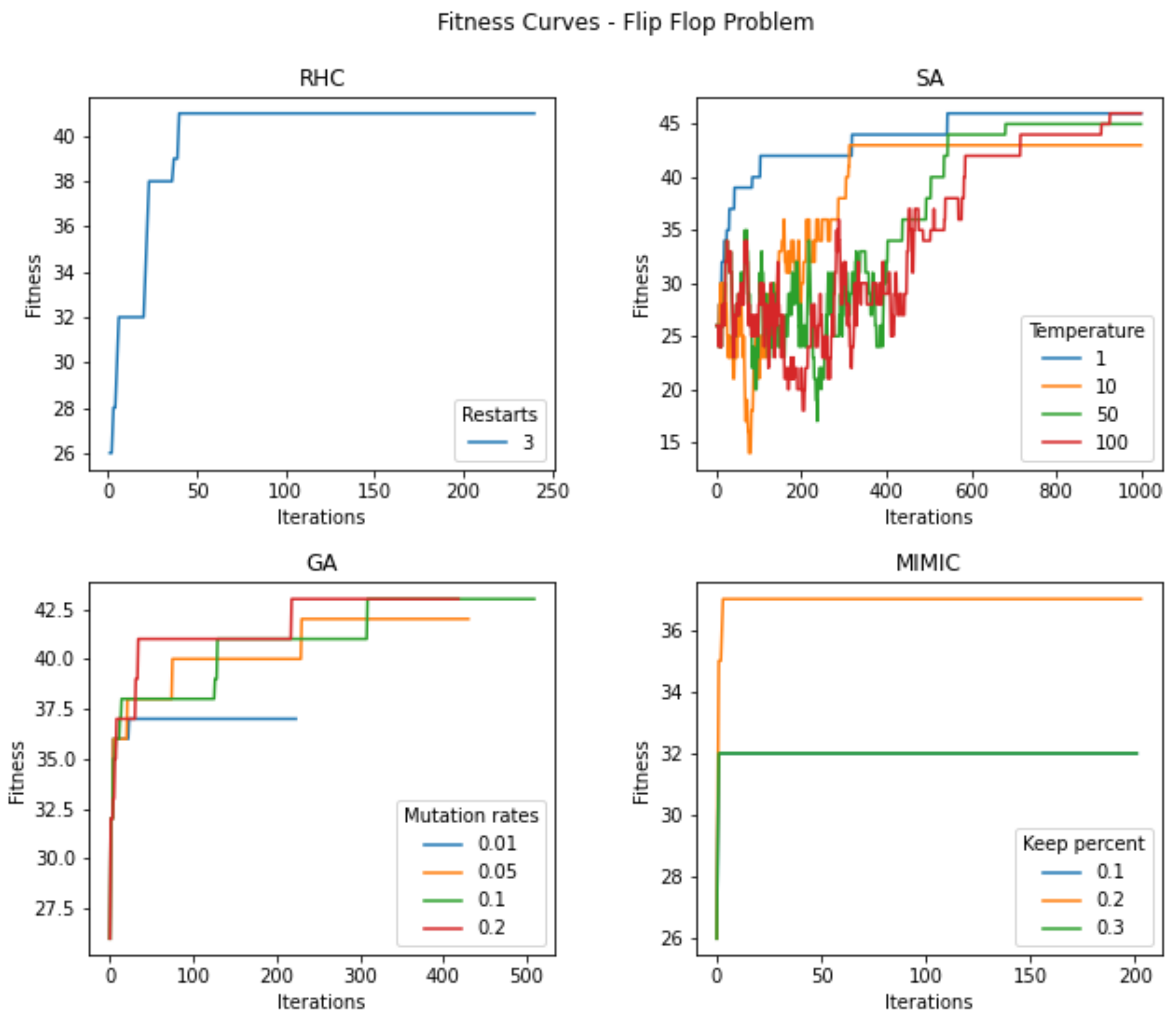


3.2 Problem 2: Flip Flop Problem (FFP)

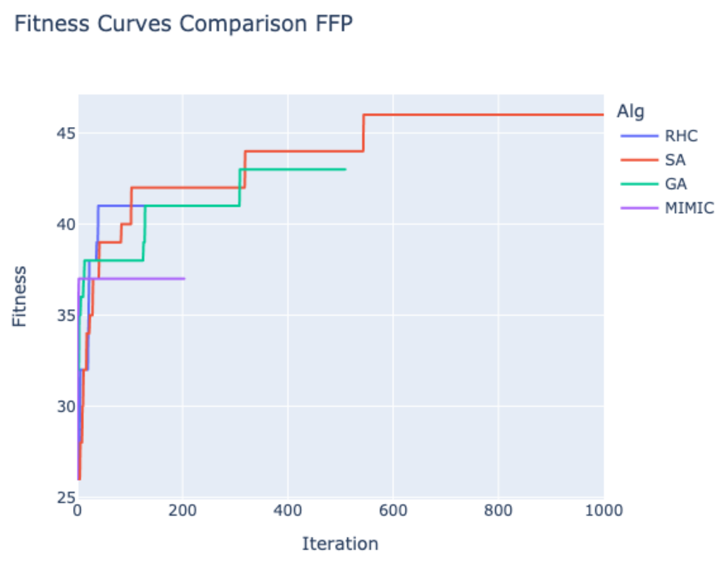
FFP is a problem that counts the number of times of bits alternation in a bit string. In binary case, consecutive digits “01” and “10” are counted as 1 in the fitness function. A maximum fitness bit string would be one that consists entirely of alternating digits. SA should excel here, since the evaluation functions are inexpensive to compute.

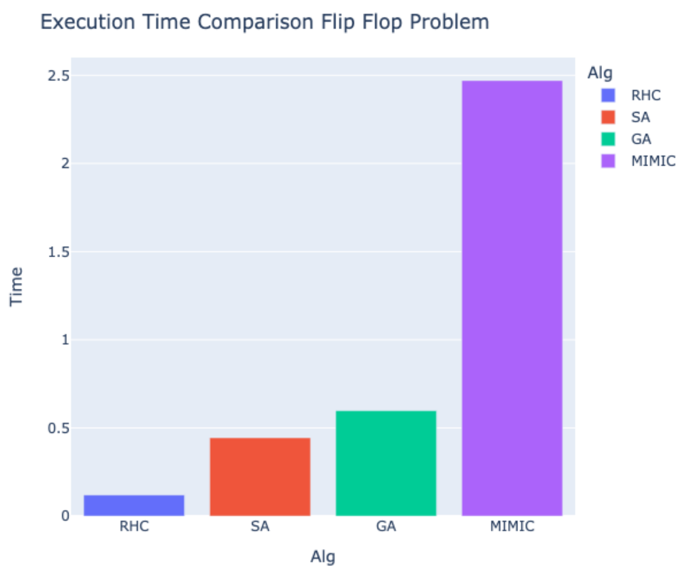
3.1.1 Results

We selected a FFP with length 50. We also tested the four algorithms with different parameters and plotted them in the figure below. Simulated annealing has temperature from 1 to 100, genetic algorithm has mutation rate from 0.01 to 0.2, and the keep percent of MIMIC ranges from 0.1 to 0.3.
For SA, the best parameter was Temp=1. Temp=100 also reached the highest fitness score, but it took more iterations. For GA, the best parameter was Mutation rate=0.2. For MIMIC, the best parameter was Keep percent=0.2. The performance for SA and GA largely depends on their parameter selection. We can see that when temperature is high, such as 100, the fitness score of SA oscillates a lot, showing that it’s jumping back and forth among the domain to search for global optima. And when mutation rate is too small (0.01), GA gets trapped in local optima and leads to low fitness score.



The 2 figures below show the fitness curves and time complexity of the 4 algorithms together. SA outperforms all other algorithms as finds the highest fitness value. Timewise, RHC took the least amount of time, followed by SA, GA, and MIMIC.





3.3 Problem 3: Knapsack Problem (KSP)

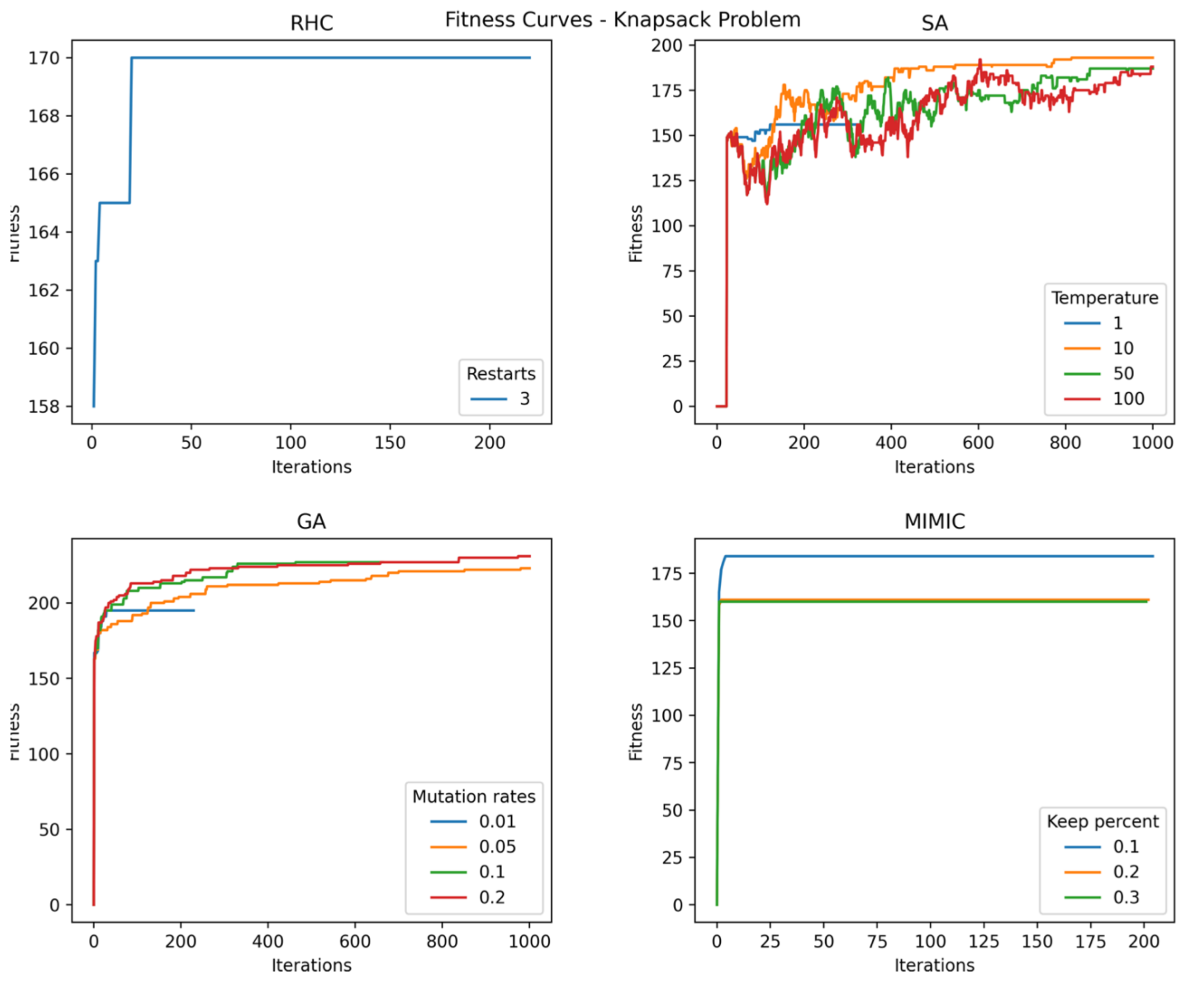
Knapsack Problem is a NP-Hard optimization problem. Given a set of items, find the items to put into the knapsack so that the total weight is less than or equal to a given limit and the total value is maximized.
The most common problem being solved is the 0-1 knapsack problem, which restricts the number x_i of copies of each kind of item to 0 or 1. Given a set of n items numbered from 1 up to n, each with a weight w_i and a value v_i, along with a maximum weight capacity W [5]. The goal is to

$$\text{maximize} \sum_{i=1}^{n} v_i\, x_i \qquad \textit{subject to} \sum_{i=1}^{n} w_i\, x_i \leq W \textit{ and } x_i \in \{0,1\}$$

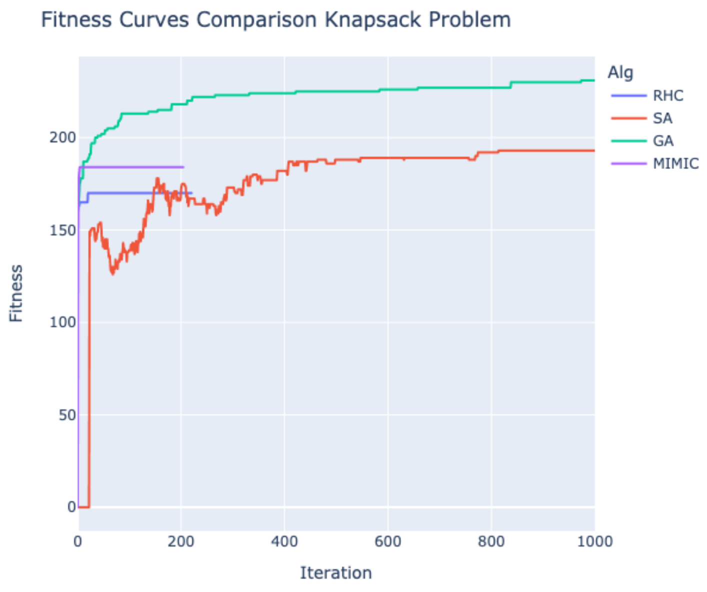
The strength of MIMIC was highlighted in this context, as it exploited the underlying structure of the problem space that was learned from previous iterations.

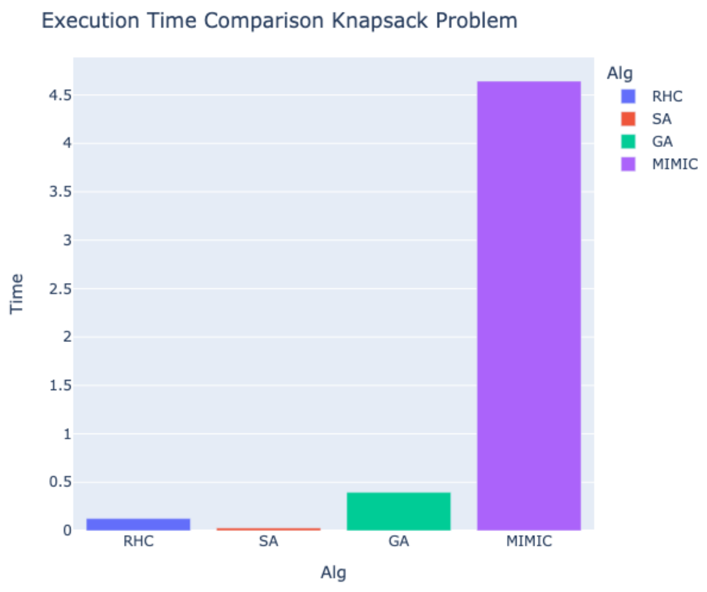
3.3.1 Results

We set the Knapsack Problem length to be 100 and again, tested the four algorithms with different parameters. The results are shown in the figure below. Simulated annealing has temperature from 1 to 100, genetic algorithm has mutation rate from 0.01 to 0.2, and the keep percent of MIMIC ranges from 0.1 to 0.3. For SA, the best parameter was Temperature=10. For GA, the best parameter was Mutation rate=0.2. For MIMIC, the best parameter was Keep percent=0.1.
This problem is more complex than both the four peaks problem and flip flop problem above. SA and GA still performed relatively well with respect to fitness scores. However, here MIMIC achieved a higher fitness score than RHC.



Again, GA outperforms other algorithms in this experiment with the highest fitness score and not much execution time. However, within 200 iterations, MIMIC actually outperformed SA and RHC. SA took many more iterations to finally reached a solution slightly better than that of MIMIC's. Looking at the number of function evaluations completed by each algorithm, MIMIC and RHC have fewer iteration than GA, but GA achieved better scores.

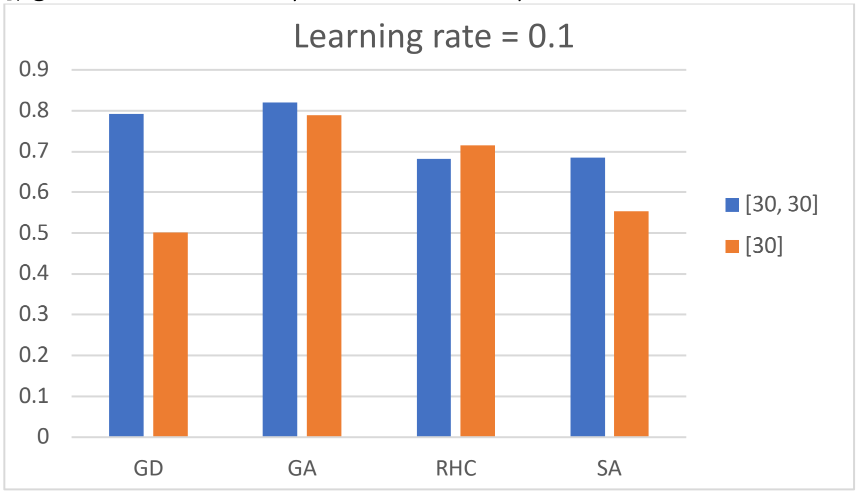




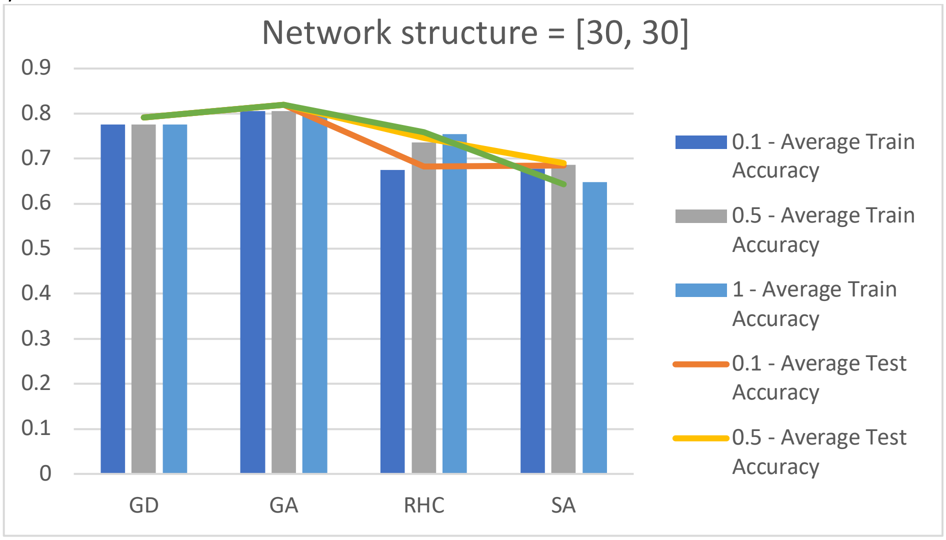
4 Neural Network Weight Optimization

The default of credit card clients Data Set from UCI Machine Learning repository was used for this portion of the analysis [6]. It's a binary classification problem with 23 features and 30K instances. The features are used to predict whether this person will default payment (Yes = 1, No = 0). All features have been standardized before sending into the algorithms. And the algorithms aim to minimize classification error.

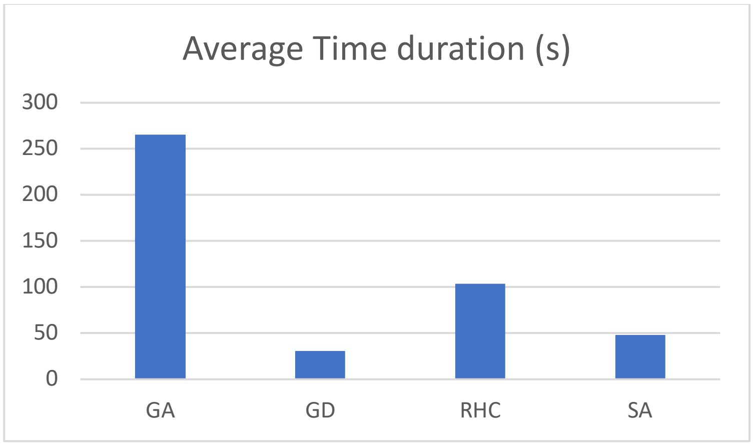
RHC, GA, SA and gradient descent are all implemented via the Python mlrose_hiive package. Activation function is "relu". 3 levels of learning rates (0.1, 0.5, 1) and 2 network structures (1 layer with 30 node [30], and 2 layers with 30 nodes each [30, 30]) are tuned to compare the algorithms on 3 different aspects: training accuracy, testing accuracy, and execution time.
For a given learning rate = 0.1, the train and test accuracy of the algorithms are shown below. GA actually outputs the highest accuracy scores, followed by gradient descent. RHC and SA have similar accuracy. Also, 2-layer structure ([30, 30]) gives better accuracy scores than 1-layer structure in most cases.



For a given network structure [30, 30], the train and test accuracy of the algorithms are shown below. GA again outputs the highest accuracy scores, followed by gradient descent, RHC and SA. Different learning rates have different affects among the algorithms. For GA and GD, the selected 0.1, 0.5, 1 learning rates didn't affect accuracy scores much. For RHC, learning rate =1 gives the highest accuracy and for SA, learning rate = 0.1 gives the highest accuracy.



However, the performance of GA comes with a price. It took much longer time than other algorithms. The execution time comparison is shown below. Gradient descent actually is quite time-saving and takes the least amount of running time.



Please refer to the overall summary in the table below.

Learning rate	Network Structure	Train Acc	Test Acc	Time duration (s)	Algorithms
0.1	[30]	0.501333	0.501167	54.459327	GD
0.1	[30]	0.706792	0.715667	88.835707	RHC
0.1	[30]	0.558333	0.554167	37.408807	SA
0.1	[30]	0.778292	0.7885	142.931549	GA
0.1	[30, 30]	0.775667	0.791333	9.507584	GD
0.1	[30, 30]	0.675333	0.6825	143.950049	RHC
0.1	[30, 30]	0.677958	0.6855	61.418447	SA
0.1	[30, 30]	0.805458	0.8195	391.79685	GA
0.5	[30]	0.501333	0.501167	52.402444	GD
0.5	[30]	0.788	0.798167	74.518705	RHC
0.5	[30]	0.703958	0.727167	36.707485	SA
0.5	[30]	0.778292	0.7885	145.844689	GA

0.5	[30, 30]	0.775667	0.791333	9.278991	GD
0.5	[30, 30]	0.736333	0.746167	129.163191	RHC
0.5	[30, 30]	0.686417	0.69	59.08548	SA
0.5	[30, 30]	0.805458	0.8195	386.431355	GA
1	[30]	0.501333	0.501167	50.52032	GD
1	[30]	0.785208	0.794667	67.460807	RHC
1	[30]	0.628583	0.644	35.330739	SA
1	[30]	0.778292	0.7885	141.447043	GA
1	[30, 30]	0.775667	0.791333	9.269156	GD
1	[30, 30]	0.753875	0.758	116.926419	RHC
1	[30, 30]	0.647375	0.643	58.186959	SA
1	[30, 30]	0.805458	0.8195	384.689235	GA

5 Conclusion

Based on the previous implementation of various problems, we can then summarize the advantages and disadvantages for each algorithm in the table shown below. For different problems, we need to consider their complexity and features to select the most suitable optimization algorithm. Also, for each algorithm, parameter tuning is necessary for finding good solutions.

Algorithms	Advantage	Disadvantage
RHC	1. Fast 2. No parameter tuning required	1. Doesn't work well for complex problems (domain has many local optima)
SA	1. Fast 2. Relatively good at finding optimal values	1. Paramter tuning can be tricky. (sometimes low temperature can give better results than high temperature)
GA	1. Almost always outputs good results, performs well on very complex problems.	1. In general is fast, but can be time-consuming for complex problems (for example, neural networks)
MIMIC	1. Good at complex problems, especially the ones with time-consuming fitness functions 2. Record the structure of underlying distribution.	1. Much slower than above algorithms 2. Doesn't always output relatively good results

6 References

[1]. Hill climbing: https://en.wikipedia.org/wiki/Hill_climbing
[2]. Optimization techniques: simulated annealing: https://towardsdatascience.com/optimization-techniques-simulated-annealing-d6a4785a1de7
[3]. Introduction to genetic algorithms: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3
[4]. Bonet, Jeremy S. De; Isbell, Charles L.; Viola, Paul (1 January 1996). "MIMIC: Finding Optima by Estimating Probability Densities". Advances in Neural Information Processing Systems: 424.
[5]. Knapsack problem: https://en.wikipedia.org/wiki/Knapsack_problem
[6]. Default of credit card clients: https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients