Randomized optimization – report

Randomized optimization – report 1
Description 1
Part 1 - Optimization problems 2
4 peaks 2
8 Queens 3
K-color 4
Part 2 – Neural networks 5
Data 5
Algorithms and params 5
RHC 5
SA 5
GA 5
Results 6
Final performance 6

Description

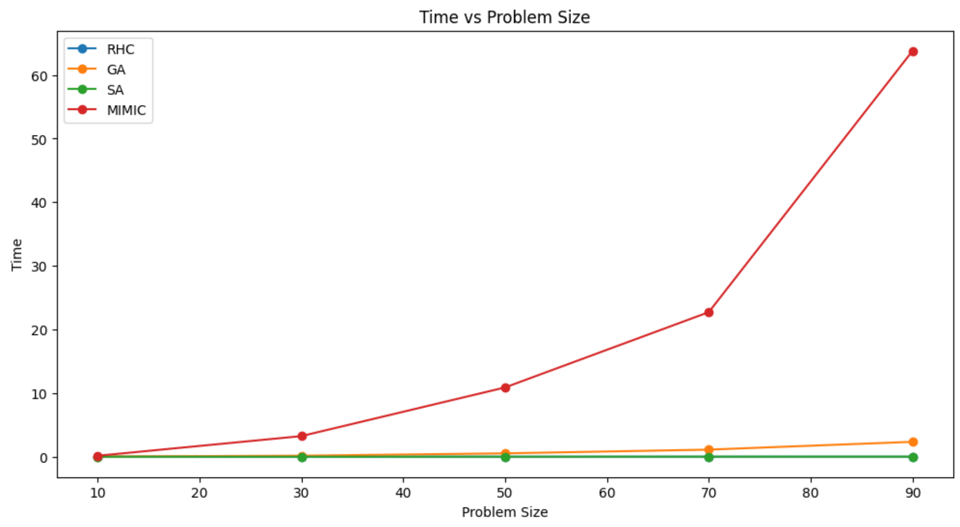
In this lab we examined 4 local random search techniques: randomized hill climbing, simulated annealing, generic algorithm and MIMIC.

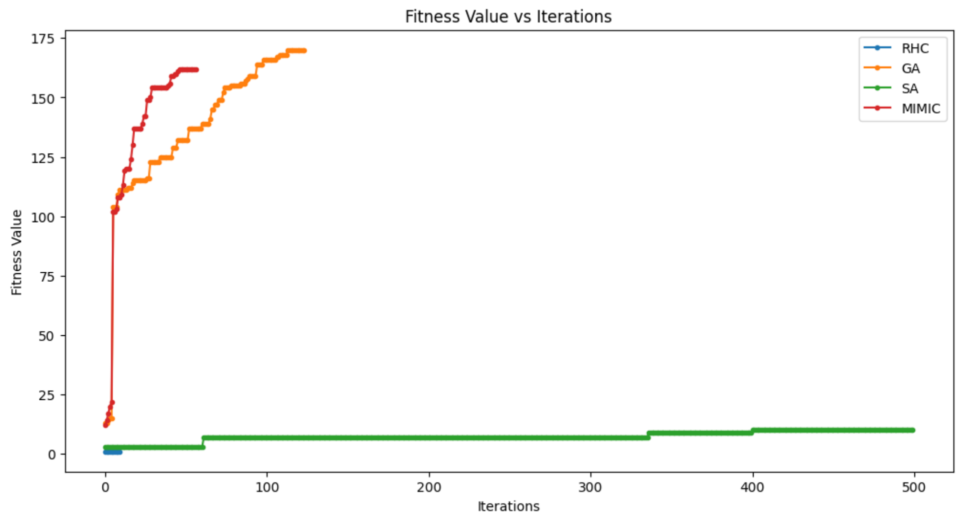
After that we applied the same algorithms to a neural network problem.

Part 1 - Optimization problems

4 peaks

The “4 peaks” is a classic optimization problem, that is commonly used as a benchmark. The problem is called 4 peaks because its data represents 4 local maximas in the function landscape. The problem is interesting, because it has local maximas and global maxima, which we need to find.

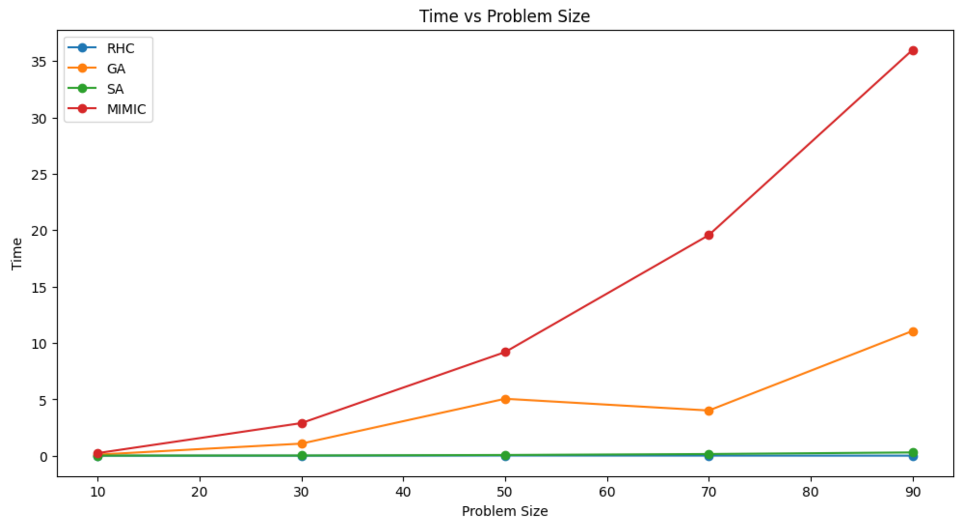


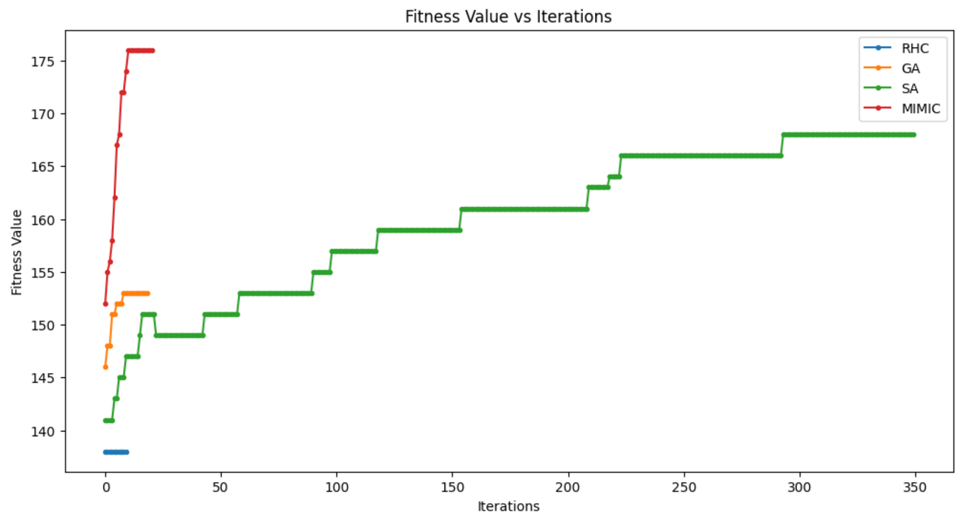


MIMIC was able to reach it’s highest fitness value with less iterations, compared to the others, but at the end of the day SA showed a bit higher score, but it required more than 2000 iterations to get to it. GA showed the second best results (after SA) but it took much fewer iterations.

8 Queens

In the 8-Queens problem, you are given a chessboard with eight queens (and no other pieces) and the aim is to place the queens on the board so that none of them can attack each other.

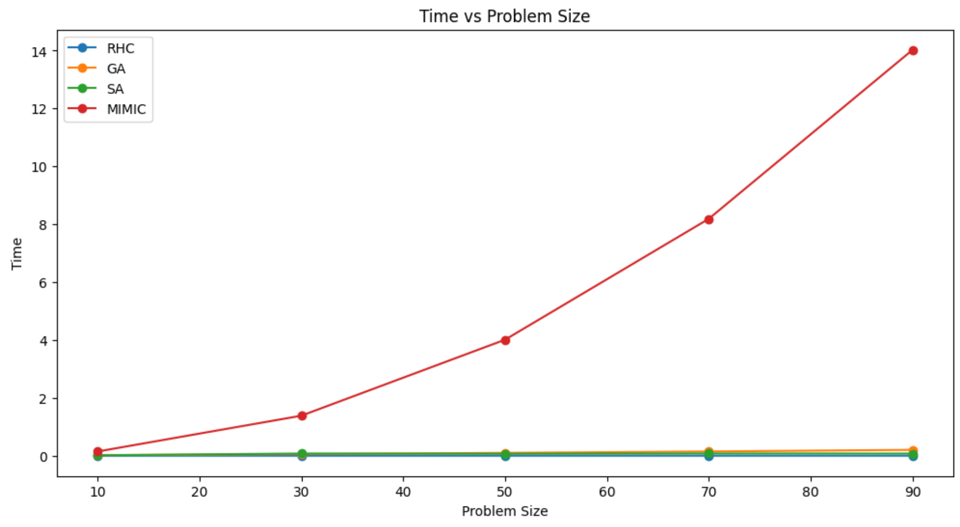


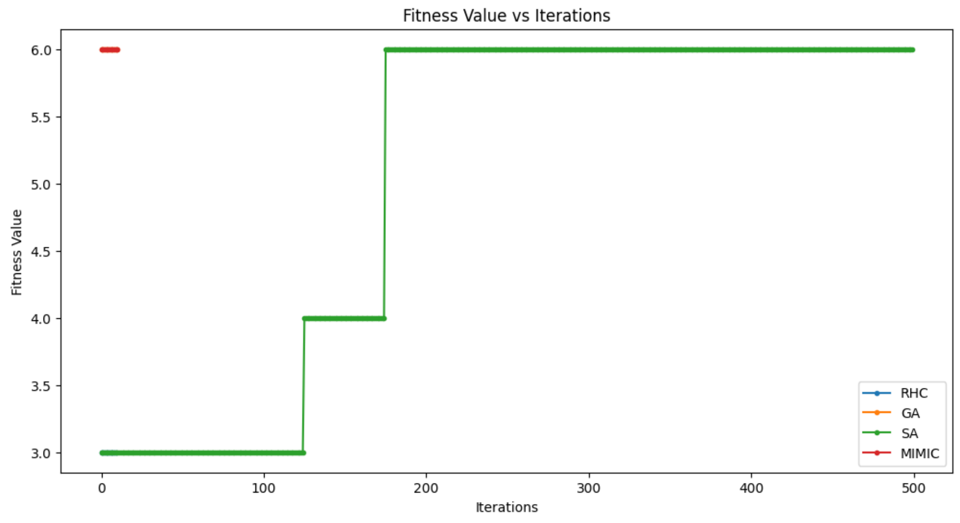


For 8 queens MIMIC showed the best results with fewer iterations. Again, SA took the most interations to reach the maximum fitness value. RHC finished last.

K-color

Evaluates the fitness of an n-dimensional state vector x = [x_0, x_1,....x_{n-1}], where x_i represents the color of node i, as the number of pairs of adjacent nodes of the same color.





As in previous problems MIMIC showed the best results with the minimum number of iterations but took more time per iteration. It became worse with the problem size increasement.

Part 2 – Neural networks

Data

This is one of the most useful problems that engineers solve in commercial organizations. In my company another department tried to identify if the transaction was a fraud or not. I will try to do the same for this learning project. Column “fraud” represents a binary classification. I had to drastically reduce dataset size, because it couldn’t finish training in appropriate time (in 12 hours it was still running for learning rate 0.1, which was the first one). Dataset is limited to 50k rows instead of 1kk.

Algorithms and params

```
learning_rates = [0.1, 0.2, 0.5, 0.7, 1]
hidden_nodes = [10, 20, 100, 200, 350, 500]
```

RHC

```
mlrose_hiive.NeuralNetwork(hidden_nodes = hidden_nodes, activation = 'relu', algorithm = 'random_hill_climb', max_iters
= 1000, bias = True, is_classifier = True, learning_rate = lr, early_stopping = True, clip_max = 5, max_attempts = 10,
random_state = random_state, restarts = 5)
```

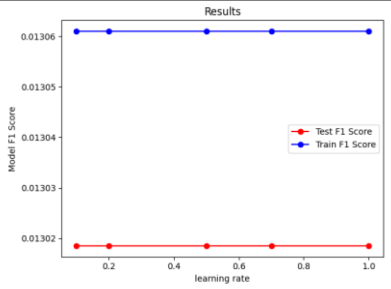
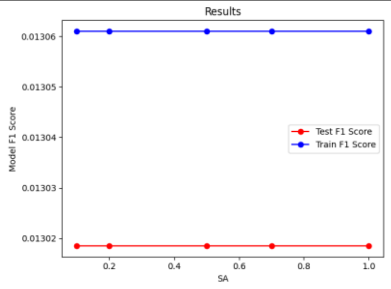
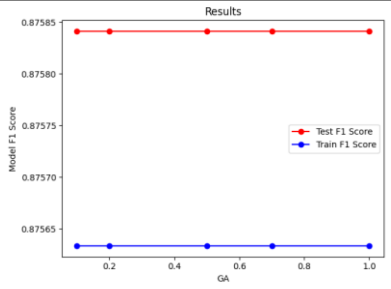
SA

```
nn_model = mlrose_hiive.NeuralNetwork(hidden_nodes = hidden_nodes, activation = 'relu', algorithm =
'simulated_annealing', max_iters = 1000, bias = True, is_classifier = True, learning_rate = lr, early_stopping = True,
clip_max = 5, max_attempts = 10, random_state = random_state)
```

GA

```
nn_model = mlrose_hiive.NeuralNetwork(hidden_nodes = hidden_nodes, activation = 'relu', algorithm = 'genetic_alg',
max_iters = 1000, bias = True, is_classifier = True, learning_rate = lr, early_stopping = True, clip_max = 5, max_attempts =
10, random_state = random_state)
```

Results

RHC	SA	GA
		

Final performance

Algorithm	Loss	Train F1 score	Test F1 score
RHC	33.01083741	0.01306104	0.01301845
SA	33.01083741	0.01306104	0.01301845
GA	0.89445693	0.87563345	0.87584133

GA took the most time, then SA. RHC was the fastest among these three.

Learning rate has no effect on model performance. At least if it's less than zero, which is common recommended range, otherwise model would learn too fast and can cause the model to converge too quickly to a suboptimal solution.

In my case the learning rate was too small, and the process got stuck.

Both RHC and SA performed much worse than GA in terms of f1 score.

I ran 1 more iteration with a learning rate of 10 to check if it makes any difference. Results are provided below:

Algorithm	Loss	Train F1 score	Test F1 score
RHC	33.01083741	0.01306104	0.01301845
SA	3.03281598	0.87563345	0.87584133
GA	0.89445693	0.87563345	0.87584133

It's hard to compare 1-1 with Assignment 1 results, because models from assignment 1 performed much faster and I was able to use 1kk rows compared to 10k here. In my case f1 was higher when I used classic MLPClassifier.

I believe it's possible to find better weights and learning rate to achieve better results, but it will take significant amount of time to train all possible combinations