1. Introduction

This paper explores the performance of 4 different randomized optimization algorithms:

- RHC - Randomized hill climbing
  A meta-algorithm built on top of the hill climbing algorithm. It iteratively does hill-climbing, each time with a random initial position. Not only optimizing from an initial condition but also exploring the space.
- SA - Simulated annealing
  Focuses more on the exploration of a solution space, by randomly choosing next-steps with some probability. This method further decreases the likelihood of getting stuck in local optima.
- GA - Genetic algorithm
  In GAs, a population (pool of possible solutions), undergo mutation and crossover, produce new generations based on fitness of prior generations. By exploiting historical information, better solutions can be "evolved" over generations.
- MIMIC - Mutual information maximizing in put clustering
  By analyzing the global structure of the optimization problem, we can perform an informed randomized search through the solution space and, in turn, to refine the estimation of the structure.

To measure and compare the performance of these algorithms, I performed four experiments:

- Four Peaks
- N Queens
- One Max
- Weight optimization for Neural Networks

The first three optimization problems are both aiming to maximize over discrete-valued parameter spaces. The last one is a continuous-state minimization problem, which are not supported in the case of MIMIC, only RHC, SA, GA are performed.

I evaluated their performance based on two factors: accuracy and time. The results are in section 2~5 respectively.

The implementations of these algorithms and fitness functions were pulled from the Mlrose library.

2. Four Peaks (GA)

2.1. Description

Given an N-dimensional input, the evaluation function for Four Peaks problem (Baluja and Caruana, 1995) is defined as:

$$f(\vec{X}, T) = \max\left[tail(0, \vec{X}), head(1, \vec{X})\right] + R(\vec{X}, T)$$

where

$$tail(b, \vec{X}) = number\ of\ trailing\ b's\ in\ \vec{X}$$

$$head(b, \vec{X}) = number\ of\ leading\ b's\ in\ \vec{X}$$

$$R(\vec{X}, T) = \begin{cases} N\ \ if\ tail(0, \vec{X}) > T\ and\ head(1, \vec{X}) > T \\ 0\ \ otherwise \end{cases}$$

There are two sharp global optima at the edge of the problem space with wide basins of attraction designed to lead "blind" optimizers toward the local optima. See Figure 1 for example. GA and MIMIC are more likely to find the maximum than RHC and SA.

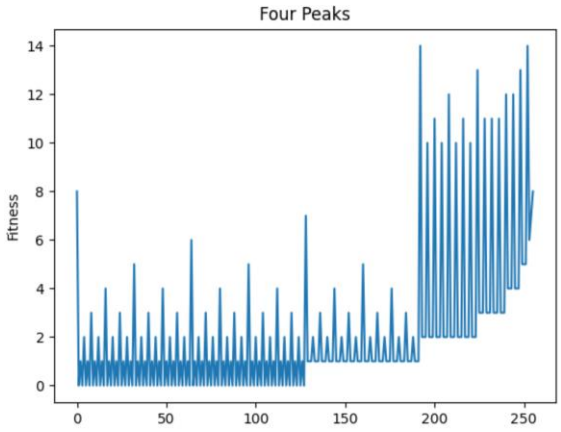


Figure 1. Four Peaks (N = 8)

The experiment was run in 2 steps: Step 1, choose a problem size N, tune the hyper-parameters for each algorithm, evaluate their convergence properties; Step 2, change problem sizes, evaluate each method's accuracy and function properties (time consumptions, number of function calls etc.).

RHC and SA started with random initial state. I didn't limit the iteration numbers for both algorithms, they stopped when they converged (may not reached global optima).

2.2. Results

The hyper-parameters I tuned:

SA:

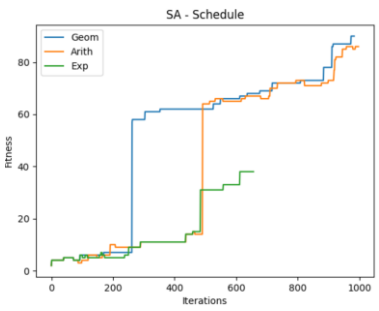
- schedule: It controls how temperature T decayed: geometrically, arithmetically or exponentially.

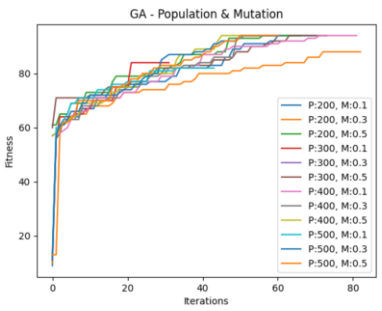
GA:

- pop_size: The size of initial population. The larger the size, the more sampling points to better understanding the structure, while the more computational expensive.
- mutation_prob: Probability of a mutation at each element of the state vector during reproduction. Stands for the ability of random exploration.

MIMIC:

- pop_size: The size of initial population.
- keep_pct: Proportion of samples to keep at each iteration of the algorithm. It affects the ability to simulate problem's probability distribution.





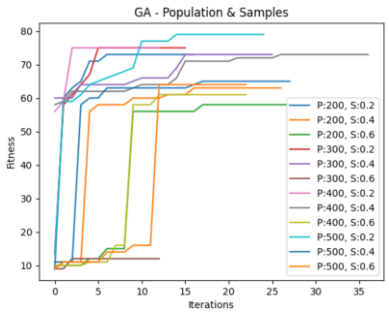
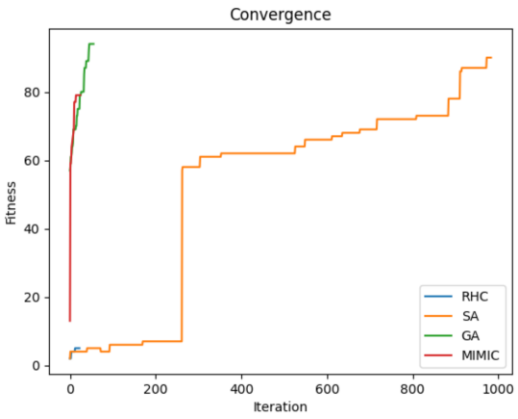


Figure 2. SA Figure 3. GA Figure 4. MIMIC

The problem size I used was 50. From the results above, I chose GeomDecay for SA, pop_size=400 and mutation_prob=0.5 for GA, pop_size=500 and keep_pct=0.2 for MIMIC. The combinations of parameters do affects the convergence significantly.



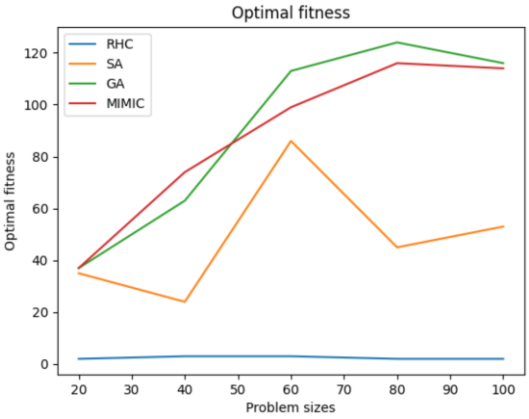
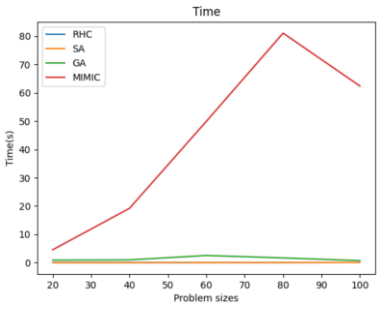
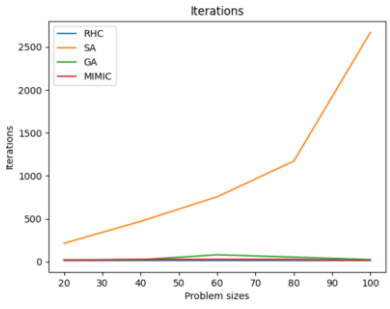


Figure 5. Figure 6.





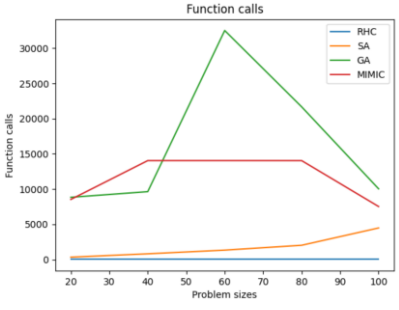


Figure 7. Figure 8. Figure 9.

From Figure 5, GA converged quickly with the best result. MIMIC in design took fewer iterations to converge, while local optima can still be a problem. SA returned a relatively good value after large number of attempts. It occasionally accepting a decreased fitness function so that it has a chance to escape from the local optimum. The “random walk” saved it from getting stuck like RHC.

I varied the problem size from 20 to 100. From Figure 6, it highlights the greedy nature of SA and RHC, even SA could not always guarantee a “good” fitness. For RHC, this problem is obviously too complex. When there is structure, GA and MIMIC's representation allow it to find better points.

From Figure 7 ~ 9, although SA always needs more iterations to converge, the time consumption for it is the smallest. On the contrary, a single iteration of MIMIC takes far more time. And GA and MIMIC makes more function calls than SA and RHC.

Improvement: Randomized optimization by its nature, takes chances. We could use the averages from multiple run to decrease the effects of randomness. There are other hyper-parameters to tune for GA, and we could tune each algorithm per problem sizes. Especially MIMIC, the performance for MIMIC largely depends on the parameter selection. The stopping criteria could also be changed. Theoretically these algorithms can finds the global optimum if they run long enough of time. Especially for SA and RHC.

3. N Queens (SA)

3.1. Description

In chess, the queen can attack any piece in the same row, column or diagonal. In the N-Queens problem, a chessboard with eight queens are given. The aim is to place the queens on the board so that none of them can attack each other (Russell and Norvig (2010).

The n-dimensional state vector:

$$x = [x_0, x_1, \dots, x_{n-1}]$$

where x_i represents the row position (between 0 and n-1, inclusive) of the 'queen' in column i.

The original fitness function is to evaluate the fitness as the number of pairs of attacking queens. In order to turn it into a maximization problem, I changed the fitness function to count the number of pairs of queens do not attack each other.

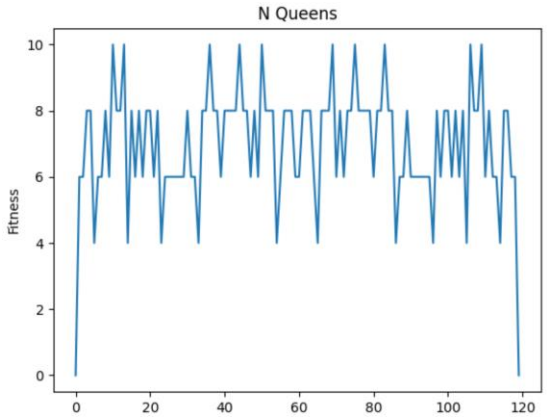


Figure 10. 5-Queens

See Figure 10 for example. There are 10 scattered global optimums with relatively narrow attraction basin. And there are plenty of not so bad local optima which can make SA and RHL achieve "good" performance even if they fail to find the optimal solution.

This experiment followed the same steps as described in section 2.1, both algorithms stopped when they converged.

3.2. Results

I tuned the same hyper-parameters as I mentioned in section 2.2.

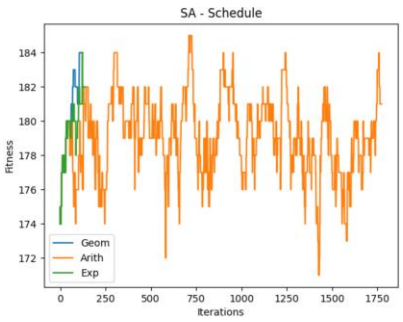


Figure 11. SA

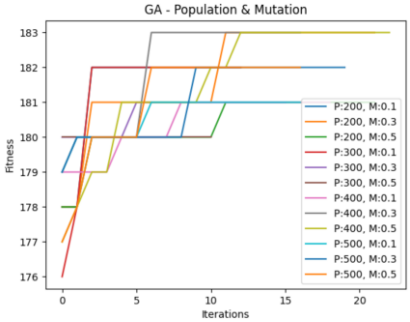


Figure 12. GA

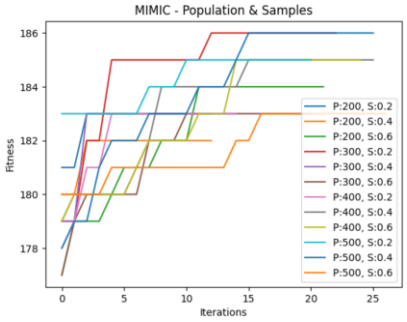


Figure 13. MIMIC

I used N=20. From the results above, I chose GeomDecay for SA, pop_size=400 and mutation_prob=0.3 for GA, pop_size=300 and keep_pct=0.2 for MIMIC.

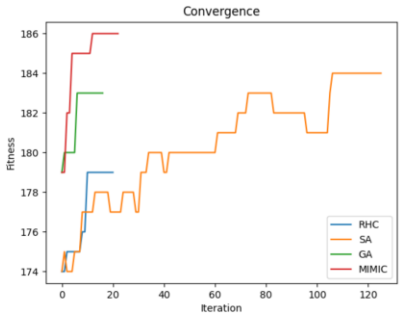


Figure 14.

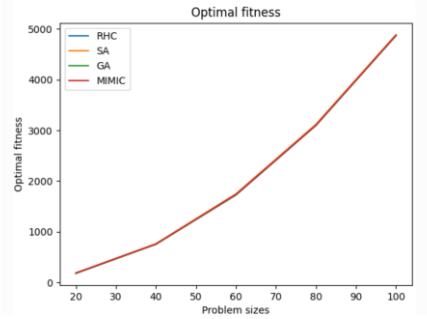


Figure 15.

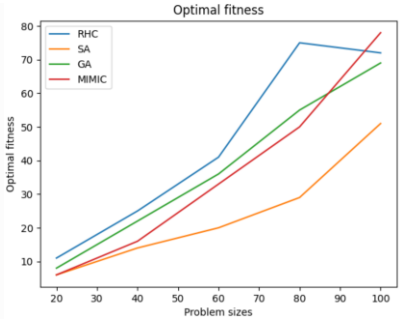


Figure 16.

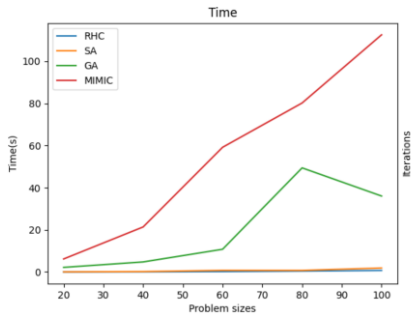


Figure 17.

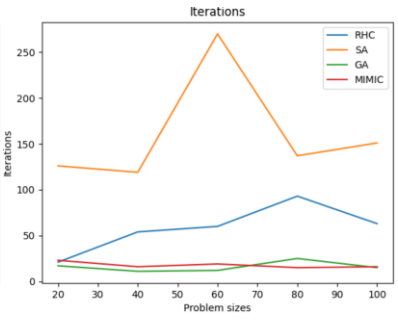


Figure 18.

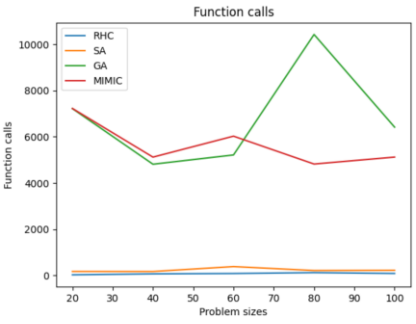


Figure 19.

To better illustrate the difference, I performed a minimization problem used the same settings as showed in Figure 16.

From Figure 14 ~ 16, both algorithms achieved similar results while neither of them can find the global maximum. SA performed better among them, which agree with our assumptions. Since GA does not gain much from mutation, and MIMIC's knowledge of past points does not truly assist in improving the

performance, this allows SA to excel. It's likely that there is no discernable structure to the problem. This problem is encoded in a way that rewards brute-force behavior. Simple approaches can also worked.

From Figure 17 ~ 19, SA and RHC uses more iterations while MIMIC and GA takes more time due to they make much more computations.

4. One Max (MIMIC)

4.1. Description

Given an N-dimensional input, the evaluation function for One Max problem is defined as:

$$Fitness(x) = \sum_{i=0}^{n-1} x_i$$

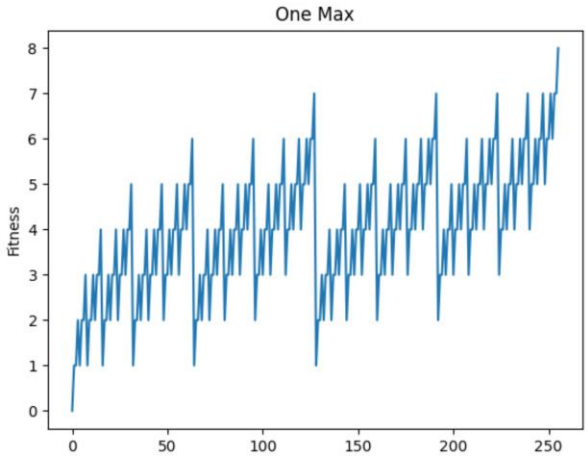


Figure 20. One Max (N = 8)

See Figure 20 for example. There is only one sharp global optimal at the edge with huge amount of local optimals which could trap SA and RHL. Algorithms that can take advantage of the structure information like MIMIC and GA have a better chance to achieve best performance.

This experiment followed the same steps as described in section 2.1, both algorithms stopped when they converged.

4.2. Results

I tuned the same hyper-parameters as I mentioned in section 2.2.

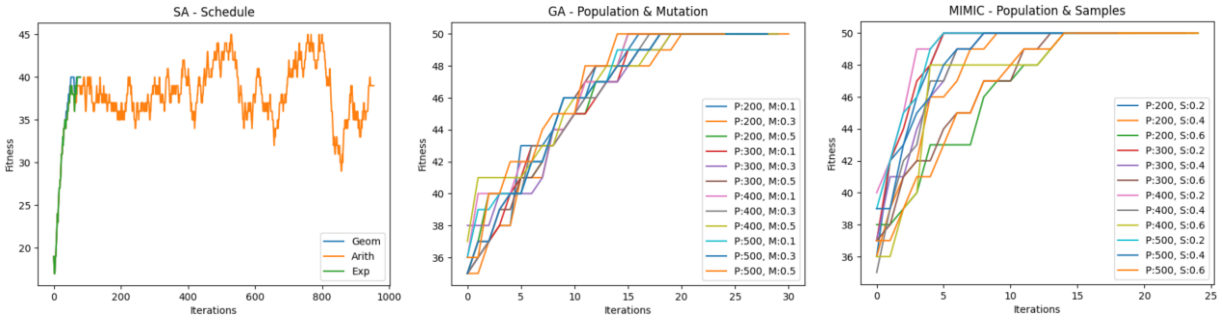
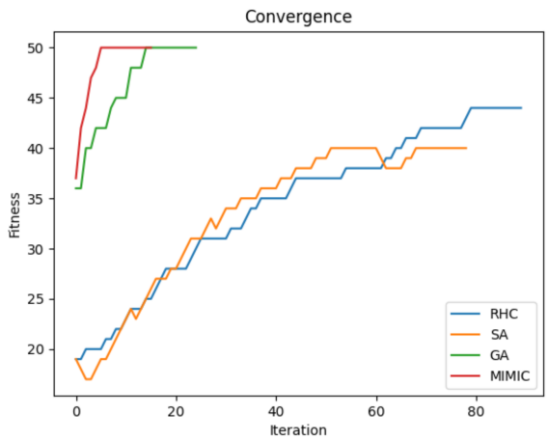


Figure 21. SA Figure 22. GA Figure 23. MIMIC

The problem size I used was 50. From the results above, I chose GeomDecay for SA, pop_size=500 and mutation_prob=0.5 for GA, pop_size=300 and keep_pct=0.2 for MIMIC.



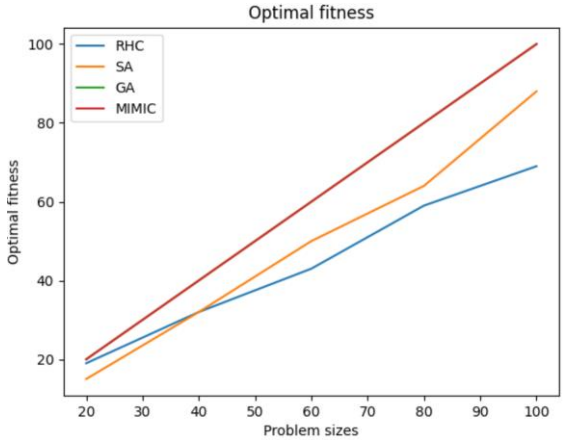
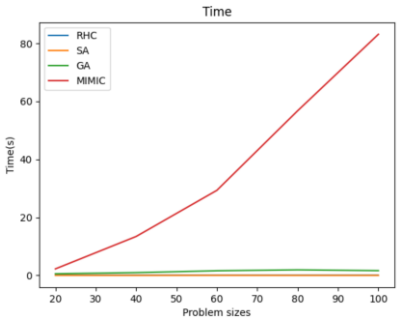
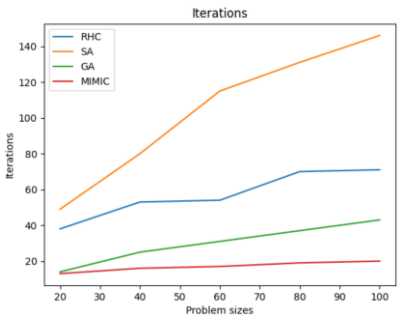


Figure 24. Figure 25.





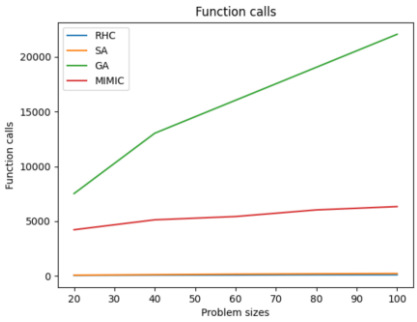


Figure 26 Figure 27. Figure 28.

From Figure 24, MIMIC converged quickly with the best result. RHC and SA can achieved a relatively good value after large number of attempts.

I varied the problem size from 20 to 100. From Figure 25, MIMIC and GA can always returned the global optima since they were able to capture the underlying structure of the problem, especially MIMIC, it can combine information from all the maxima. When there is structure, GA and MIMIC's representation allow it to find better points.

From Figure 7 ~ 9, although SA and RHC always needs more iterations to converge, their computations are cheaper. MIMIC's processing time largely increases with the problem sizes.

5. Weight optimization for Neural Networks

5.1. Description

The weights of the NNs are usually updated using Backpropagation. But one deficiency of Gradient descent is that it may stuck at local optima. I replaced it with RHC, SA and GA to see how they performed.

The dataset I used is red variant of the Portuguese "Vinho Verde" wine appraisal samples. The goal is to model wine quality based on physicochemical tests. There are 1599 samples with 11 features. The target is quality scoring between 0 and 10 (only 3 to 8 samples in this dataset). In order to make it a binary-classification problem, I turned the 10 point scale into dichotome variable: quality that 6 or higher as good/1, and the rest as not good/0.

I randomly split the dataset with 0.2 of it as test set and keep the same distribution in both sets. I implemented the Neural networks with 1 hidden layer and 10 nodes in it.

I tuned the hyper-parameters for each algorithms on training set, evaluated their converge properties, and compared the accuracy and time consumptions in the following section.

5.1. Results

I used the 5-fold Grid search CV to tune the hyper-parameters and chose the combinations with the largest mean accuracy score. As so, we can fit the data without overcommit to it:

GD:

- learning_rate

```
Best parameters {'nn__hidden_nodes': (10,), 'nn__learning_rate': 0.001}
Best score 0.7458823529411764
```

RHC:

- learning_rate: step size, the larger the size, the quicker the convergence.

```
Best parameters {'nn__learning_rate': 3}
Best score 0.6927481617647059
```

SA:

- learning_rate: step size
- schedule: It controls how temperature T decayed: geometrically, arithmetically or exponentially.

```
Best parameters {'nn__learning_rate': 46, 'nn__schedule': GeomDecay(init_temp=1.0, decay=0.99, min_temp=0.001)}
Best score 0.6724080882352942
```

GA:

- learning_rate: step size
- pop_size: The size of initial population.
- mutation_prob: Probability of a mutation at each element of the state vector during reproduction.

```
Best parameters {'nn__learning_rate': 1, 'nn__mutation_prob': 0.1, 'nn__pop_size': 200}
Best score 0.7372824754901961
```

From the results above, GD and GA fit the training set better than RHC and SA.

Then I run each models for 10000 iterations to train the dataset. The convergence curves for each algorithm are shown in Figure 29. As I expected, RHC converged much quicker than SA. To my surprise, RHC achieved the best fitness while GA returned the worst after huge amount of time.

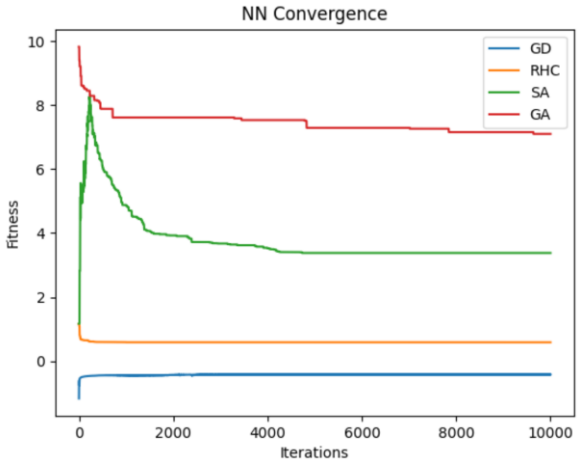


Figure 29.

	CV score	training score	test score	F1 score	time (s)
GD	0.75	0.79	0.76	0.79	37
RHC	0.69	0.72	0.72	0.73	14
SA	0.67	0.74	0.7	0.73	17
GA	0.74	0.79	0.73	0.79	3296

After fitting the dataset, I made predictions for both training and test sets and used the accuracy scores and F1 scores to evaluate their performance. From the chart above, GD and GA out performed RHC and SA, while they took longer training time instead, especially GA. It's a trade-off between time consumption and accuracy. And they overfit the training data a little bit than RHC and SA. Although RHC achieved the lowest fitness value, its prediction scores are not good enough. It's likely that fitness is not an indicator for training accuracy.

Overall, GD is the best algorithm for this problem with the highest test score and reasonable training time.

6. Conclusions

The performance for randomized optimization algorithms is sensitive to input representation. For problems where retaining history would assist in creating a better optimization, MIMIC or GA would be a better performing algorithm. However, when the optimal point is random over a wide range of values, RHC and SA may be the better solution.

References

De Bonet, JS., Isbell C, and Viola P (1997). MIMIC: Finding Optima by Estimating Probability Densities. Massachusetts Institute of Technology