CS7641 Fall 2022 HW2 Randomized Optimization

Fung Yi Yuen
fyuen3@gatech.edu

***Abstract*— Four randomized optimization algorithms (randomized hill climbing, simulated annealing, generic algorithm, and MIMIC) were implemented on three optimization problems (flip flop, four peaks, and continuous peaks) to find the fittest solution. Then all the algorithms (except MIMIC) were used to find the best weights for a neural network.**

I. RANDOMIZED OPTIMIZATION (RO) ALGORITHMS

1.1. Randomized Hill Climbing (RHC)

RHC is a hill climbing mathematical technique but with randomly selected initial states. It starts with a random arbitrary solution and then searches for a better solution stepwise until no further improvement. Its goal is to find the maximum or minimum value in a target function through many iterations.

1.2. Simulated Annealing (SA)

SA is a stochastic global search algorithm. Unlike RHC which only accepts a better solution in each search, SA accepts a worse solution in an attempt to avoid being stuck at local optima. Similar to annealing a metal, SA has a parameter T indicating the temperature of its algorithm. The higher the T, the more likely SA will accept a worse solution. The temperature will gradually cool down based on a cooling schedule as the number of iterations increases. When T becomes very small, SA acts like hill climbing to find the nearest optimal value.

1.3. Genetic Algorithm (GA)

GA is a type of evolutionary algorithm which uses a heuristic global search technique. By evaluating each solution in the population, the fittest ones are selected to breed (crossover or mutate) the next generation and this reproduction process keeps going on until an acceptable fitness level has been reached or the number of generations reached its maximum.

1.4. Mutual-Information-Maximizing Input Clustering (MIMIC)

MIMIC is an optimization algorithm which passes information from one iteration to the next. It first randomly samples from the regions of input space where a cost function C() is most likely to be optimal. Then it uses an effective density estimator to capture the structure on the input space (Isbell). MIMIC keeps estimating new distributions whose fitness is at least better than threshold θ until θ reaches maximum.

II. OPTIMIZATION PROBLEMS

2.1. Four Peaks (GA-friendly)

Four peaks problem is designed to have 4 optimal solutions in which 2 are local optima with large attractive basins whereas 2 are global optima at the edges of the fitness population. This problem was created by Baluja and Caruana to be GA-friendly (Balujia and Caruana, 1995). The fitness function is defined as follows:

$z(x)$ = Number of contiguous Zeros ending in Position 100

$o(x)$ = Number of contiguous Ones starting in Position 1

$$REWARD = \begin{cases} 100 & \text{if } o(x) > T \wedge z(x) > T \\ 0 & \text{else} \end{cases}$$

$$f(x) = MAX(o(x), z(x)) + REWARD$$

If T=10 and the bit string is in the length of 100, the best score is 189 because the fittest solution is a bit string which scores REWARD=100 and consists of either 11 or 89 leading 1's with all zero bits follow behind, i.e. the number of 1's and 0's > threshold T, and either one of the following combination is true:

# of leading 1's	# of trailing 0's	Total # of bits
11	89	100
89	11	100

Thus two global optima within four peaks.

This problem would be interesting to show how the 4 algorithms behave differently. As T becomes larger, the attractive basin for the 2 local optima will also be larger. RHC and SA would be stuck at local optima with the large basin. MIMIC might still find the fittest solution if the problem size is small. GA would crossover and mutate the elitist until the population converges to similar fitness level. Thus GA is supposed to be able to find the best solution irrespective of the sample size.

2.2. Continuous Peaks (SA-friendly)

Continuous peaks is an extension of four peaks but consists of many local optima. Unlike four peaks which the global optima are located at the two ends of a fitness function, continuous peaks global optima is in between its many local optima, i.e. the T+1 trailing 0 or 1 can be located anywhere in a bit string. The problem's structure can hardly be exploited by GA through crossover or mutation. RHC can easily get stuck in local optima. SA or MIMIC would be good at exploring the underlying structure to find the global optima of this problem.

This problem is interesting as it can be a comparison of four peaks. Optimization algorithms perform differently when there are continuous peaks instead of four peaks. It can highlight the strength and weakness of different algorithms.

2.3. Flip-flop (MIMIC-friendly)

The flip-flop problem counts the number of alternating bits in a bit string. For example, a bit string "1010" scores 3 since it has 3 alternations. "100" scores 1 since it flips one bit. The fittest bit string with length N (N-bit) is the one with N-1 consecutive alternating bites i.e. 1010101010…

This is a simple yet interesting problem because each bit depends on its next bit which is similar to pattern matching and it requires information from previous samples to search for a better solution. MIMIC is expected to perform well at this problem because it is good at communicating information from one iteration to the next.

III. METHODOLOGY

mlrose-hive library was used to call the optimization algorithms and to solved to three discrete optimization problems. To ensure fair comparison among algorithms, maximum iteration was set to 3000, range of problem size was set to 20 - 100, maximum number of attempts was set to 100 in each problem for each algorithm.

Parameters which are specific to algorithms was listed as below:

- SA: cooling schedule uses geometric decay with init_temp=1.0, decay=0.99, min_temp=0.001
- GA: mutation probability = 10%
- MIMIC: 200 samples per iteration, proportion of samples kept per iteration = 20%

IV. FOUR PEAKS

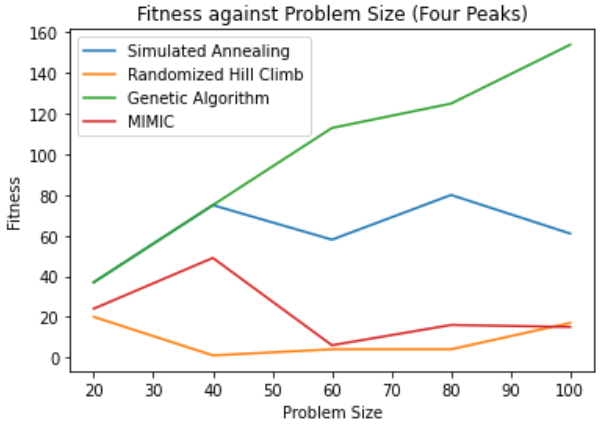


Fig.1 Fitness vs problem size (Four Peaks)

From fig.1, GA consistently performs better than other three algorithms with different problem sizes.

RHC performs the worst because it searches for a better solution at each step so it easily gets stuck at the large attractive basin of the two local optima.

SA almost performs as good as GA when the problem size is small (e.g. 40) because it accepts temporarily worse solutions in an attempt to explore the solution space. But as problem size grows larger, the attractive basin of local optima becomes larger. Probability for SA to get stuck in large attractive basins also becomes larger. Thus SA performance drops as problem size increases.

MIMIC performs no better than GA because this problem is not a pattern finding problem and passing previous information to the next iteration is not necessarily helpful in search for the fittest solution.

GA's beats three algorithms because it exploits the solution space through crossover and mutating the elitists until the population converges to an acceptable fitness level. It will not be stuck at local optima like RHC and SA nor requires probability calculation to find the best dependency tree which might end up not representative to the cost function of this problem.

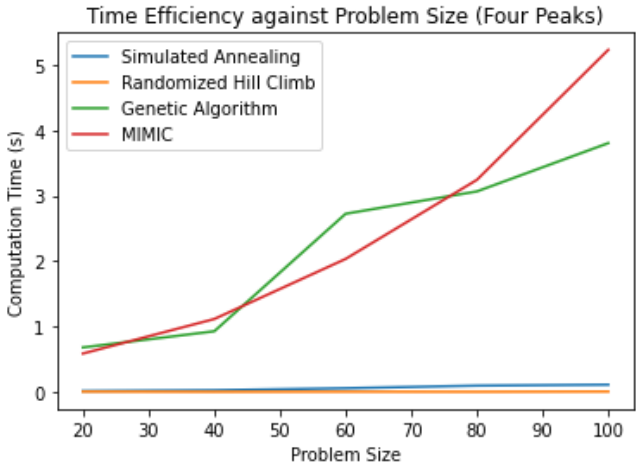


Fig.2 Compute time vs problem size (Four Peaks)

In terms of computation time (fig.2), both RHC and SA are more or less the same. They only need a small amount of time no matter how big the problem size is. It is because RHC itself is a simple algorithm which uses a single run to search for better solutions. No complex function evaluations per iteration thus it is fast to run.

SA is slightly more complex than RHC due to the random probability introduced into its algorithm (the temperature). But as the temperature cools down, it behaves like RHC which explains why its compute time is similar to RHC.

Both MIMIC and GA take more compute time than RHC and SA. MIMIC's compute time increases exponentially with problem size because significant time is needed to construct a dependency tree. As problem size grows larger, more time is required.

GA needs to evaluate the fitness of the population before mating or mutating. As the population size grows, more time is required for GA to evaluate and find the fittest samples to crossover or mutate. Thus GA's computation time increases with problem size.

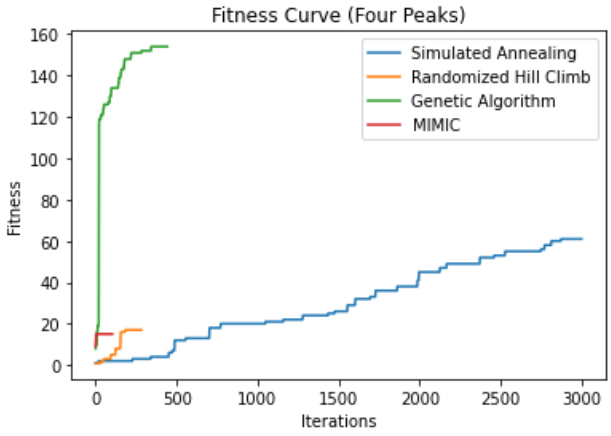


Fig.3 Fitness vs iterations (Four Peaks)

In terms of iterations (fig.3), both MIMIC and RHC stop searching at a few hundred iterations. GA reaches the highest score at around 500 iterations. SA is still on its way to reach the highest scores even after 3000 iterations.

RHC only needs a few hundreds of iterations but scores badly because it accepts the local optimal solution due to its greedy search nature. It fails to cross the large attractive basin in four peaks.

MIMIC also stops at a few hundreds of iterations. It would be due to the solutions it finds to construct a dependency tree are mostly the local optimal solutions from large attractive basins. Thus the mutual information shared between each feature and its parent are local optimal solutions, which explain the low fitness.

SA fitness score keeps increasing as number of iterations grows but still have not reached the highest score as GA does even 3000 iterations was used. It makes sense that exploring solution space which has large attractive basins takes many steps. If infinite iterations were allowed and the cooling schedule was slow enough, SA would still reach the fittest solution.

GA performs the best as it reaches the highest fitness scores within a finite number of iterations. Its mating and mutating approach helps to propagate optimal solutions per iteration thus reaching the fittest solution quickly.

IV. CONTINUOUS PEAKS

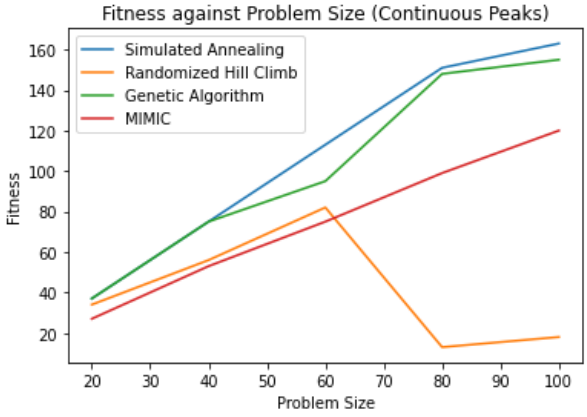


Fig.4 Fitness vs problem size (Continuous Peaks)

From fig.4, SA performs the best with different problem sizes. In fact, both SA and GA perform well at this problem. MIMIC is mediocre while RHC is the worst as problem size increases.

RHC performs the worst because of the same reason in the four peaks problem — it resorted to a local optima among many continuous peaks.

MIMIC performs better than RHC could because the relationships between peaks give information to find the highest peak, i.e. communicating previous knowledge helps MIMIC to search a better solution in each iteration. But it still can get stuck in local optima when the probability distribution of all possible solutions better than threshold θ cannot be represented by fitness function.

GA performs almost as good as SA because of its crossover and mutation nature. But the propagation process might not always generate the fittest children. Two peaks crossover can reproduce a child which is trough. Thus unlike the four-peaks problem, population convergence is not guaranteed.

SA performs the best due to its random exploring nature. When searching through many peaks, it accepts worse solutions in order to cross many small peaks to find the highest peak. Unlike the four-peaks problem, continuous peaks do not put the best solution at the very edge of a fitness function nor have a large attractive basin to trap SA at local optima. Thus SA can still roll across the solution space to find the fittest solution.

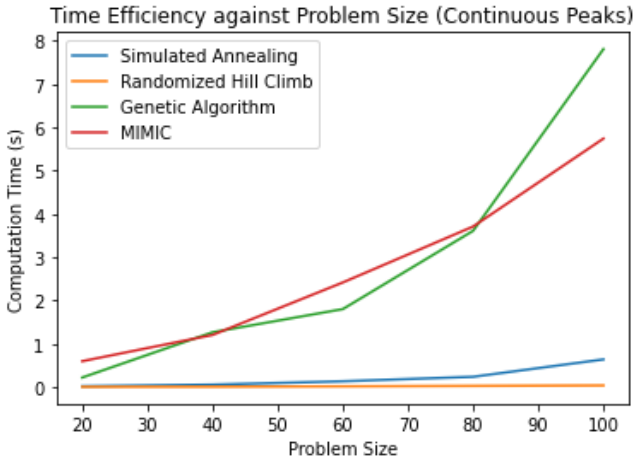


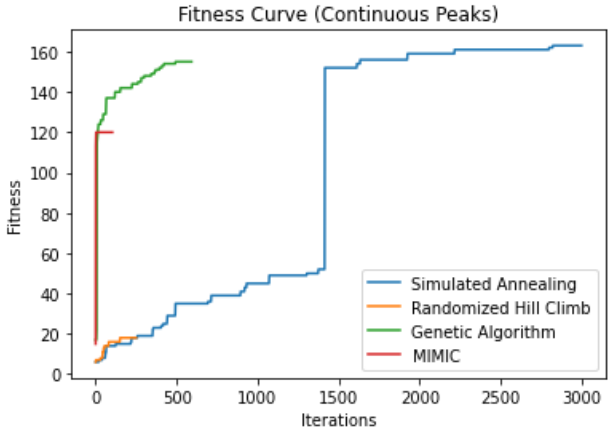
Fig.5 Compute time vs problem size (Continuous Peaks)

In terms of compute time (fig.5), the graph looks similar to the one in four-peaks problem. RHC took the least amount of time irrespective of the problem size due to the simple nature of its algorithm.

SA took slightly more time than RHC due to random exploration which is introduced by an acceptance probability function.

Both MIMIC and GA took a significant amount of computation time as problem size grows. MIMIC takes time to construct a dependency tree (the larger the problem size, the more complex the tree is). GA takes time to evaluate the fitness of the population in order to mate or mutate the next

generation (larger population size indicates longer time to evaluate the fitness).



Fitness vs iterations (Continuous Peaks)

From fig.6, GA seems the best as it reaches high fitness within fewer number of iterations than SA. But as more iterations goes, SA reaches a higher fitness score than GA.

Similar to the four-peaks problem, GA performs well because it only propagates the fittest solutions in every iteration.

SA's random search causes the slow increase in fitness even after 1000 iterations. At iteration=1500, it somehow hits the best solution which makes a sharp increase in fitness. Such a sharp increase is due to the design of continuous peaks (the global optima located among many local optima). SA might get a chance to hit the global optima before it cools down otherwise it will get stuck at local optima like RHC (the larger than problem size, the more easily SA will resort to local optima).

RHC is the worst since it resorted to a low fitness score within less than 500 iterations, which indicates it sticks around local optima in every iteration.

Unlike four-peaks where MIMIC got stuck at local optima, this time MIMIC scores quick high with a few hundreds of iterations, though not as high as GA/SA. It suggests that pattern recognition and previous information help MIMIC find better solutions in this problem to a certain extent.

V. FLIP-FLOP

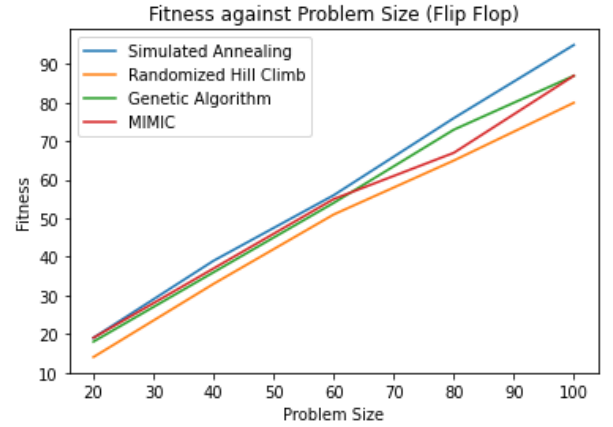


Fig.7 Fitness vs problem size (Flip flop)

From fig.7, the four algorithms perform similarly well as the problem size increases. There is a small difference at the size=100 where SA performs slightly better than the other 3 algorithms. Out of curisorty, larger problem sizes was also tested and the graph looks like below:

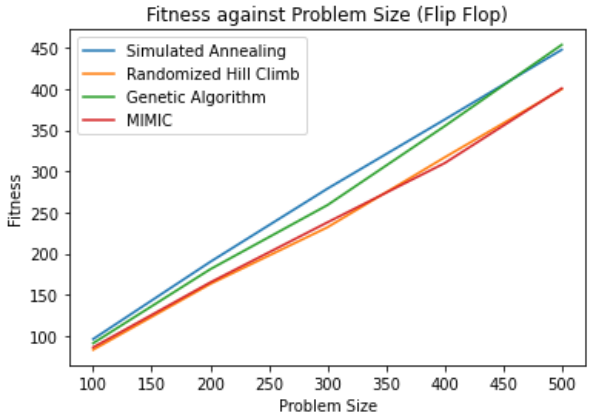


Fig.8 Fitness vs Larger problem size (Flip flop)

Turns out SA and GA are only slightly better than MIMIC and RHC in this problem. It could be due to the problem's simplicity that finding a string with all alternating bits is easy. There is no large attractive basin in the fitness function to trap SA / RHC or a global optimal hidden among many small peaks to make GA / MIMIC difficult to search for better solutions. Thus problem size does not really affect any algorithm in this problem.

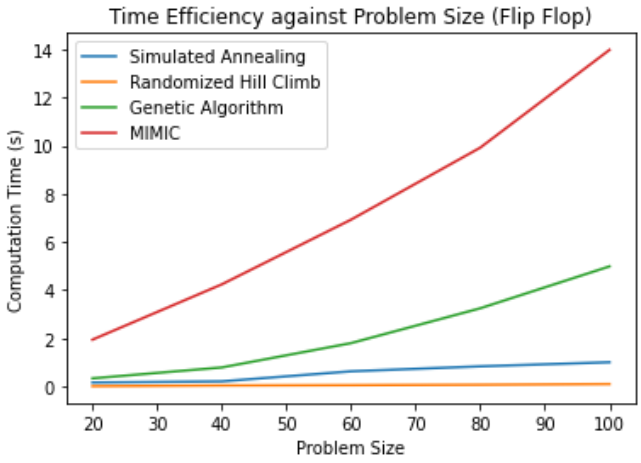


Fig.9 Compute time vs problem size (Flip flop)

In terms of compute time (fig.9), the results are consistent with the previous two problems. MIMIC took the longest time to compute due to its algorithm's complexity. The larger the problem size, the longer the compute time. GA took the second longest time. GA's compute time also increases with problem size but not as much as MIMIC. SA took slightly longer than RHC as problem size increased.

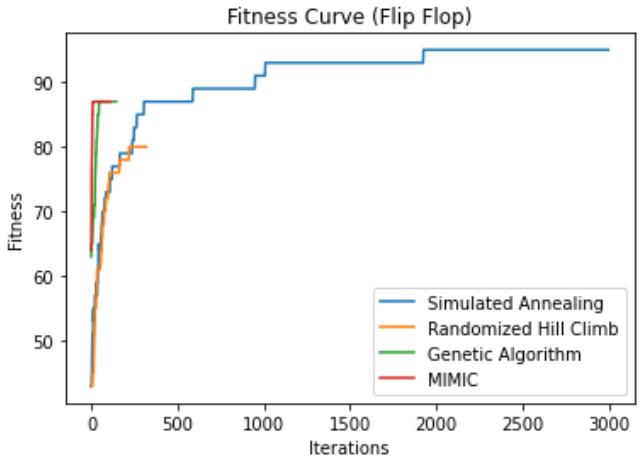


Fig.10 Fitness vs iterations (Flip flop)

From fig.10, MIMIC requires the least number of iterations to reach fitness ~90. It is because MIMIC is good at handling structure. It passes previous information to the next iteration such that it uses fewer steps to reach high fitness than any other algorithms.

GA performs similarly well as MIMIC but with a few more iterations to reach the same fitness.

SA can reach the same fitness as GA and MIMIC but with a lot more iterations. Due to its random exploring nature, it can even reach a higher fitness score as the number of iterations increases. But such a small increase in fitness takes a lot more iterations which might not be cost effective.

RHC has the lowest fitness score and takes more iterations to reach the same level of fitness of other algorithms. Due to its heuristic nature, it always resorted to local optima.

VI. SUMMARY

Certain algorithms are more suited to solve certain problems. GA works well on the four-peaks problem. SA works well on the continuous-peaks problem. MIMIC works well on the flip-flop problem.

In summary:

Algo	Suitable for
RHC	solving large instances in a few seconds, if sub-optimal solutions were accepted
SA	problems with large discrete configuration space e.g. continuous peaks
GA	discrete, stochastic, nondifferentiable or highly-nonlinear solution space
MIMIC	problems require pattern recognition or previous knowledge communication

VII. NEURAL NETWORK OPTIMIZATION

7.1. Introduction

In assignment 1 (HW1), backward propagation was used to find the best weights of a neural network. In this paper, RHC, SA and GA were used to find the best weights of the neural network trained in HW1.

7.2. Dataset

The Divorce Predictor dataset was chosen as the classification problem. It is a balanced dataset with binary classification, divorced and married. The least 10 correlated features were chosen out of 53 features to train the neural network in order to avoid "curse of dimensionality".

7.3. Preparation

Train-test split is 80:20. The training set was then further splitted into training and validation sets by ratio 80/20. F1-score was used to evaluate each algorithm's performance. mlrose-hive library was used in this experiment. A neural network with one hidden layer 16 nodes were used for different algorithms to find the best weights.

Different learning rates (0.0001, 0.001, 0.01, 0.1, 1) were fed into the neural network for each algorithm in order to find the best f1-score that an optimization algorithm can achieve. The best model found was then tested with validation and testing data.

7.4. Back propagation (serve as a benchmark)

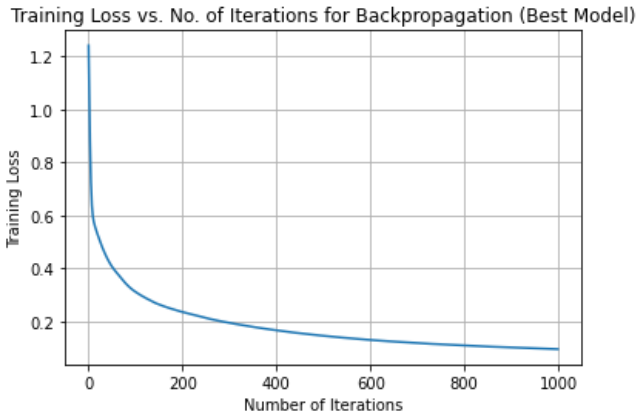


Fig.11 Training loss vs # of iterations (RHC)

Backward propagation is a weight tuning approach by reducing the sum of square error at the output end and then propagating backward through n–layers. The aim is to minimize the sum of square error through every iteration such that the best weight of a neural network can be found.

The loss curve drops exponentially and reaches a stable low value around 1000 iterations. The best f1-score obtained from the test set is 0.9116883116883117 which serves as a benchmark to see how other three optimization algorithms perform.

7.5. RHC

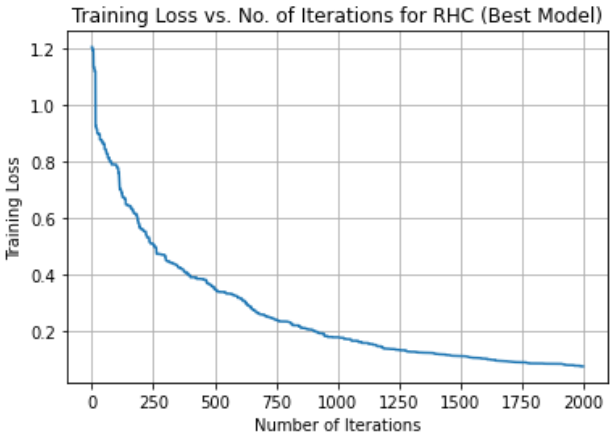


Fig.12 Training loss vs # of iterations (RHC)

RHC needs 2000 iterations to reach stable low training loss. Its loss curve looks similar to back-propagation but drops at a slower rate. It could be due to the random starts used by RHC which slows down the error loss per iterations.

The best f1-score from the test set is 0.9110723626852659, which is very close to back-propagation.

7.6. SA

Fig.13 Training loss vs # of iterations (SA)

SA needs 2000 iterations to reach low loss but the curve is not as smooth as RHC / back-propagation. It would be due to SA's random exploring nature. It accepts worse solutions when temperature T is high. That's why the beginning when the number of iterations is small, the loss curve is fluctuating. It means the error is not consistently decreasing with the number of iterations like RHC does.

But as the number of iterations goes higher, SA's temperature is cooling down which makes it searching like RHC. Thus the loss curve looks similar to RHC when the number of iterations is large (>1000).

The best f1-score from the test set is 0.8823529411764706, which is lower than RHC and back-propagation.

7.7. GA

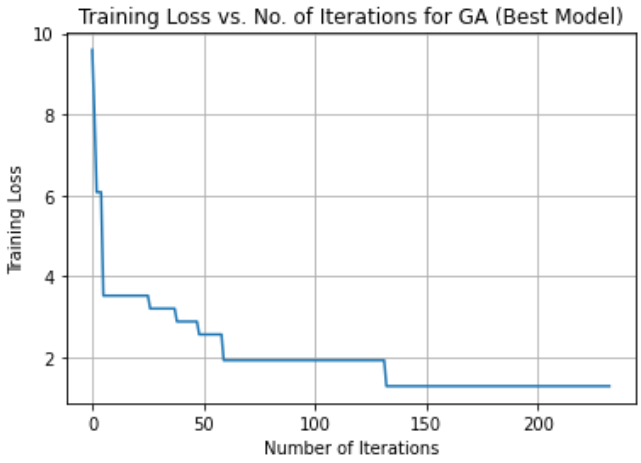


Fig.14 Training loss vs # of iterations (GA)

GA uses the least amount of iterations among the three optimization algorithms. It only needs 150 iterations to reach a stable low loss value. It drops sharply at the beginning when the number of iterations is small, which means GA finds the best weights of the neural network in much fewer steps than RHC, SA and back-propagation.

Similar to the performance in previous optimization problems' section, GA always has quick convergence within relatively fewer iterations than other algorithms. It could be due to its mating and mutating nature. It picks the fittest solutions to generate the next population every iteration thus fewer steps might be needed than gradient descent to find the optimal solution.

The best f1-score from the test set is 0.9110723626852659, which is the same as RHC and nearly the same as back-propagation.

7.8. Comparison - Scores

Best score on testing set for all optimization algorithms

Optimization approach	f1-score
Back-propagation	0.9116883116883117
RHC	0.9110723626852659
SA	0.8823529411764706
GA	0.9110723626852659

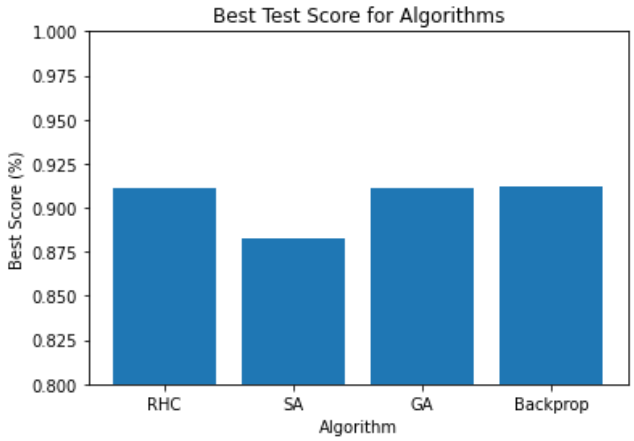


Fig.14 Best test score for all 4 algorithms

From fig.14, SA performs the worst while RHC, GA and back-propagation perform more or less the same on the testing data. Looking at the training score and validation score of SA (fig.15, fig.16), its training score is the lowest (same as GA) while the validation score is 100%, which means SA's neural network is not overfitting the training data.

But why does SA perform well on the validation set (scores 100%) but poorly on the testing set (scores 88%)? This could be due to the few samples in the dataset for validation since the dataset only has 170 entries. After the 80/20 train-test split and then further 80/20 testing-validation split, not many samples were left in the validation set. The optimized neural network is then validated by only a few samples which might not be representative to the samples in the outside world.

RHC scores the best in the training set and validation set, which means its neural network predicts well on unseen data. Thus its testing score is high as expected.

Both GA's testing and validation scores are the lowest (same as SA) but its testing score is as high as RHC, which is interesting. It means both GA's and RHC's neural networks work equally well on the testing set. It could be due to the small testing samples given which is not enough to distinguish the performance of the two algorithms.

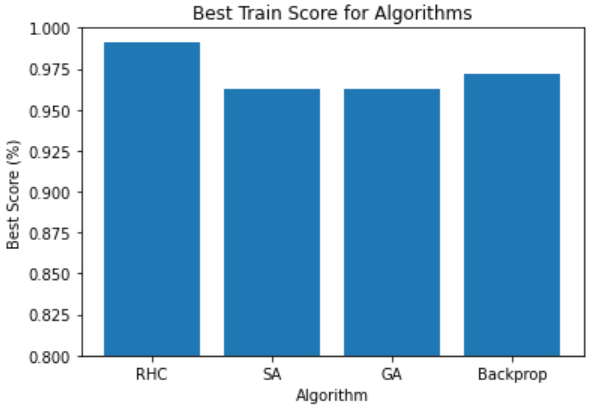


Fig.15 Best training score for all 4 algorithms

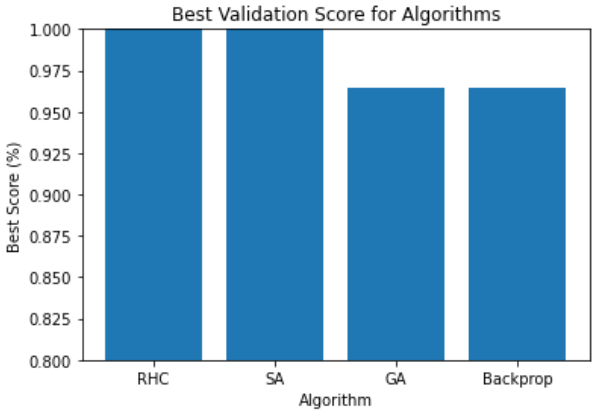


Fig.16 Best training score for all 4 algorithms

7.9. Comparison - Compute time

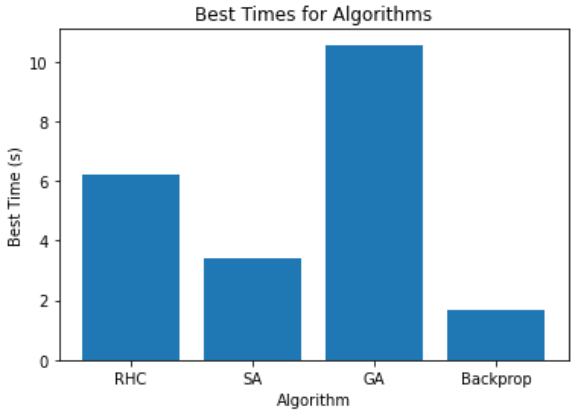


Fig.17 Best compute time score for all 4 algorithms

In terms of compute time, GA is the most time consuming due to its algorithm's complexity. It needs to evaluate fitness of the population every iteration in order to populate the fittest solutions in the next step.

Back-propagation is the fastest because minimizing the sum of square error is relatively straightforward compared to other algorithms.

RHC comes the second highest because there are many random restarts for the algorithm to search in the solution space which takes more time then SA.

SA picks one starting point and then uses high temperature to explore the solution space such that it can cross local optima and find the global optima as iterations grows. Its algorithm includes a random probability calculation which takes more time to compute than back-propagation.

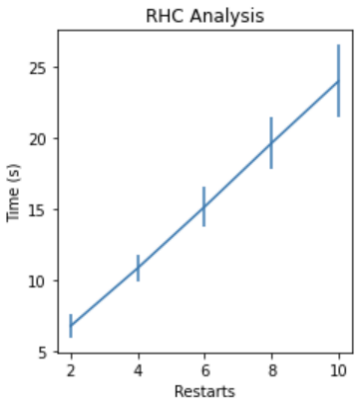


Fig.18 Time vs # of restarts (RHC)

Varying each algorithm's parameter can also affect its computation time to find the optimum weights.

Fig 18 shows RHC compute time increases linearly with the number of random restarts. More restarts means RHC explores more opportunities to find the peak but more time will be needed as a trade-off.

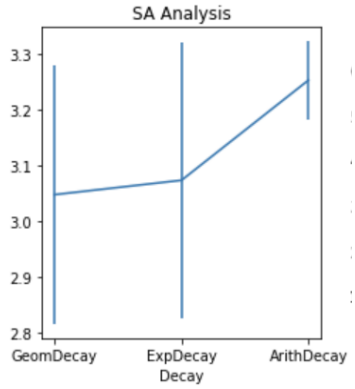


Fig.19 Time vs different cooling schedules (SA)

Fig.19 shows different cooling schedules (GeomDecay, ExpDecay and ArithDecay) of SA. It means how fast SA cools down from high temperature. Different cooling schedules are suited for different optimization problems which need to be tested when solving an optimization problem.

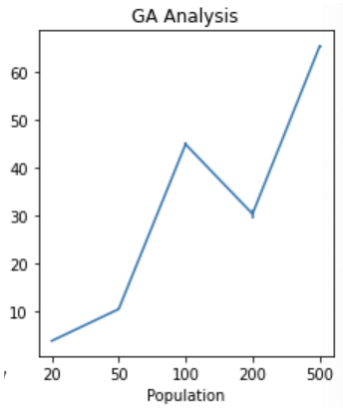


Fig.20 Time vs population size (GA)

Fig.20 shows GA's analysis time versus population size. Theoretically, the larger the population size, the more time required for analysis. But practically, if some crossovers or mutations of the elitists can produce the fittest solution, the time required to find the fittest would be less. That's why a drop in time at population size=200.

7.10. Comparison - training and validation curve

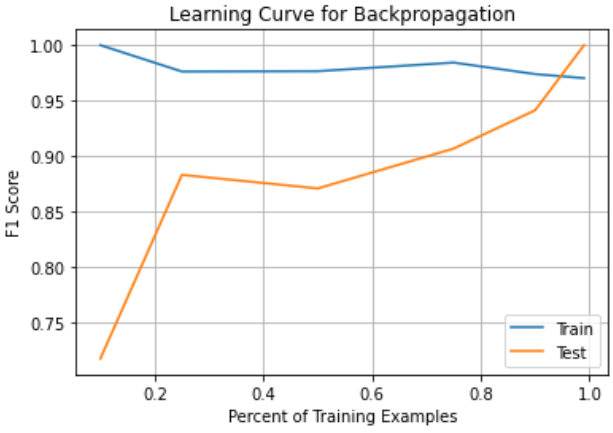


Fig.21 Learning curve for back-propagation

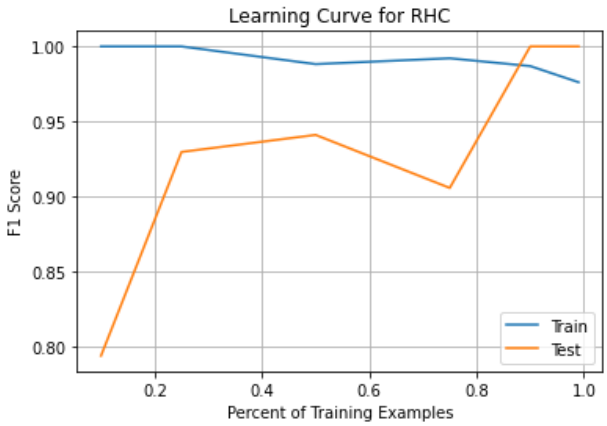


Fig.22 Learning curve for RHC

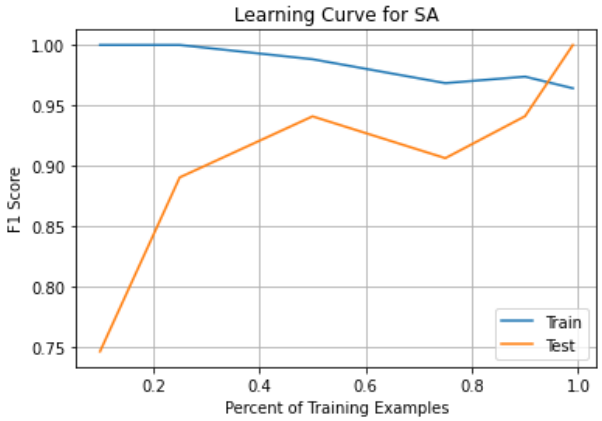


Fig.23 Learning curve for SA

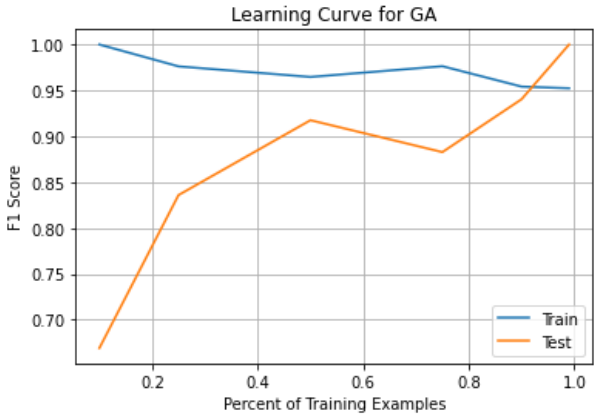


Fig.24 Learning curve for GA

Training and testing curves were plotted for all the optimization algorithms used in this experiment (fig.21, 22, 23, 24).

In general, all the training curves start at the perfect score 1 and then drop slightly as the number of training samples increases. There is a big gap between the training and testing curves at the beginning when the number of samples is small. It is because the neural networks trained have high variance. The model only fits to the small samples but fails to predict unseen data.

As more training samples are given to train the neural networks, the model starts to generalize, which is indicated by the small decrease in training score but large increase in testing score. The training score drops a bit and then plateaus while the testing score keeps increasing to meet the training curve, i.e. they converge at high score. It means the model predicts unseen data more accurately with more training data.

Notice that the testing scores in all algorithms becomes higher than the training scores and even reaches 1 when 100% training samples were given to train the model. It is due to the design of the train-test split. When fewer samples were given to train the model, more samples were reserved to test the model. Similarly, when more samples were given to train the model, fewer samples were left to test the model.

test_sizes = [0.9, 0.75, 0.5, 0.25, 0.1, 0.01]

training_sizes = [0.1, 0.25, 0.5, 0.75, 0.9, 0.99]

Above the array listed above, only 1% samples were left for the model test. It means the model trained well on 99% data and it can easily predict that 1% sample correctly. Thus testing scores are higher than training scores for all optimization algorithms.

VIII. CONCLUSION

Among the three optimization algorithms (RHC, SA, GA), both RHC's and GA's best f1-scores are as high as back-propagation. In terms of computation time, RHC is faster than GA due to its algorithm's simplicity. Although SA computes faster than RHC, its neural network could not predict testing data as accurately as RHC could. Thus RHC is the best in optimizing weights of the neural network for this classification problem.

However, if back-propagation also counts, it would be the best algorithm among the four since it consumes the least amount of compute time while scores the highest f1-score.

It is worth noting that there are many factors that affect the performance of each optimization algorithm. The problem itself such as problem size, solution space, number of hidden layers etc. The algorithm itself such as learning rate, population size, number of random starts, cooling schedules etc. The methodology itself such as train-test split ratio, time vs accuracy to rate the performance of each algorithm etc. Plotting graphs would help to observe some features of the problem as well as the algorithm for latter adjustment.

IX. REFERENCE

[1]	Isbell, C. (1996). Dr. Randomized Local Search as Successive Estimation of Probability Densities. https://faculty.cc.gatech.edu/~isbell/tutorials/mimic-tutorial.pdf
[2]	Baluja, S., & Caruana, R. (1995). https://www.ri.cmu.edu/pub_files/pub2/baluja_shumeet_1995_1/baluja_shumeet_1995_1.pdf
[3]	Mitchell, T. (1997). Machine Learning. McGraw-Hill Education.