Project 2 Analysis report

Yu Feng

GTID: yfeng40

GitHub: https://github.com/leonyufeng/CS7641_project_2

Abstract

This analysis report is to compare the performance of four Random Optimization algorithms (Random Hill Climbing, Simulated Annealing, Genetic Algorithm and MIMIC) on 3 unique optimization issues and also compare the performance on neuron network weight optimization with normal Gradient Decent. Interesting result shows Genetic Algorithm can get better fitness for neuron network weight optimization than Gradient Decent.

1. Introduction

Random optimization (RO) algorithms are widely used to find optimal value for functions and in optimization problems that are not continuous or not differentiable. In this analysis, we will use 3 different issues to study the performance and advantages of four Random Optimization algorithms: Random Hill Climbing, Simulated Annealing, Genetic Algorithm and MIMIC.

1.1 Random Hill Climbing (RHC)

Assign random start points and exploring near optimal points and moving to direction that has better fitness on each iteration.

1.2 Simulated Annealing (SA)

With some random start points, give a temperature setting that affects the probability of jumping to next points, jump to better fitness point with 1.0 probability. This increases the chance of making points jump out of local optimal.

1.3 Genetic Algorithm (GA)

Start with random parents start points. With better fitness points, adding mutation and crossover to generate child points. And then keeps on evolving like natural species survival. The eventually survived points will be more likely to approaching the optimal.

1.4 Mutual Information Maximizing Input Clustering (MIMIC)

MIMIC is trying to use sampling to find out the structure of the problem space. It using sampling to estimate the parameters of the distribution for current iteration, then generate more data with current distribution, then keep only top percentile of the sample data to estimate the parameters of the distribution for next iteration.

1.5 libraries

To speed up the analysis, open-source libraries are used in the experiments. The used libraries are mlrose[1] and mlrose_hiive[2]. mlrose is a library designed for this course and mlrose_hiive is an improvement over mlrose library. mlrose_hiive is more used in this report due to its convenient data generator, fitness functions and runner class.
The analysis includes two major parts:

1) Compare RHC, SA, GA and MIMIC on Continuous Peaks, Flip Flop and Max K Color issues
2) Use RO algorithms to optimize neuron network model weights.

2. RO algorithm performance comparison

To compare the features and performance of the four algorithms, three issues are chosen as suggested by the instruction to show case. The three issues are Continuous Peaks, Flip Flop and Max K Color.

2.1 Continuous Peaks Problem (CPP)

CCP is to find the optimized peek points in the curve/space, among them there could be multiple real global peaks and multiple local peaks. The RO algorithm is supposed to find the global peaks and try to jump out of local peaks.
The problem size is the number of points in the curve. The problem size impact to all four algorithms is shown in Figure 1. It shows with problem size increasing for Continuous Peaks Problem, RHC fitness declines with problem

size increasing. SA and MIMIC fitness score also decline with problem size increase. Interestingly, GA fitness score increase with problem size increase. It shows the nature of GA that creates species across the solution space has the potential to solve more complex problems.

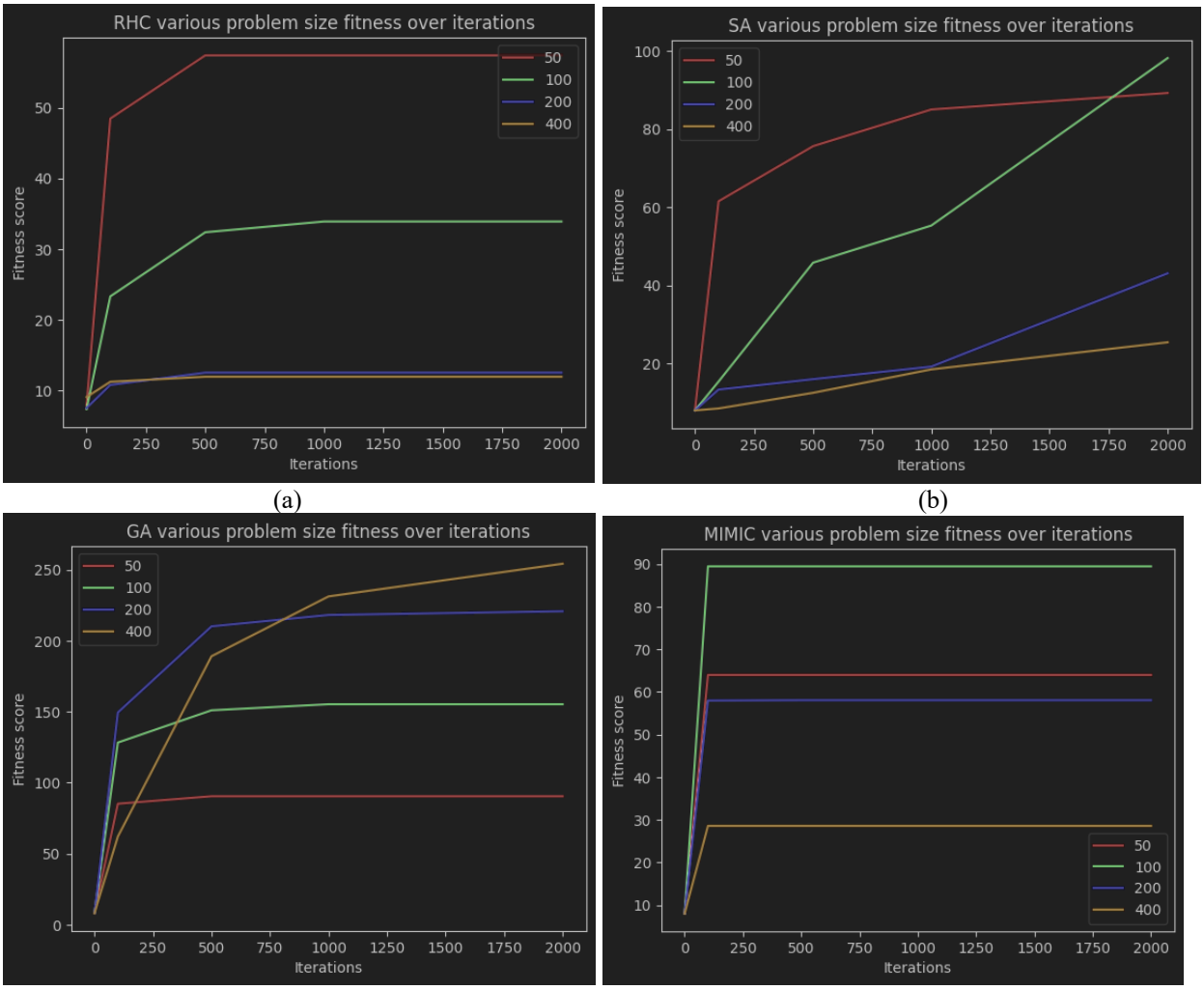


Figure 1. CPP four algorithms various problem size fitness comparison over iterations

After comparison the impact of problem size to four algorithms, problem size 100 is chosen to do further analysis.

2.1.1 RHC models

As Figure 1(a) shows, RHC reached best fitness at around 1000 iterations for all 50 to 400 problem size. RHC fitness on smaller problem size is significantly better than larger problem size.

2.1.2 SA models

After chosen problem size 100, we can compare the impact of Temperature Decay methods and initial temperature. The compared 3 Temperature Decay methods are Exponential Decay, Arithmetical Decay and Geometrical Decay. The results are shown in Figure 2. It shows Exponential Decay fitness score performed better with lower iterations for CPP.

Then from Figure 3. various initial start temperature fitness comparison, it shows higher start temperature leads to higher fitness score for CPP.

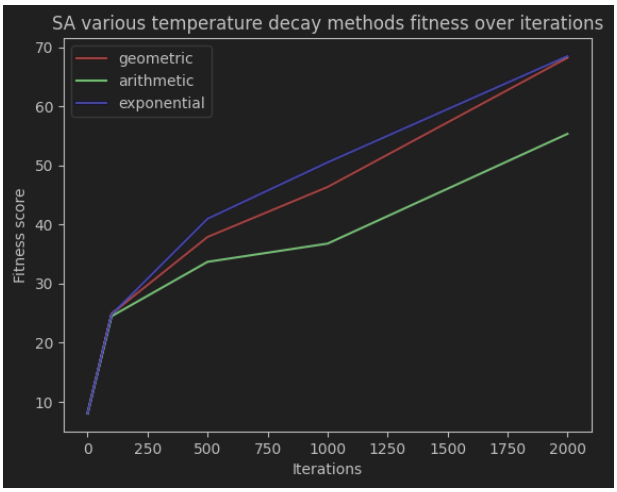


Fig 2. CPP SA temperature decay over iterations.

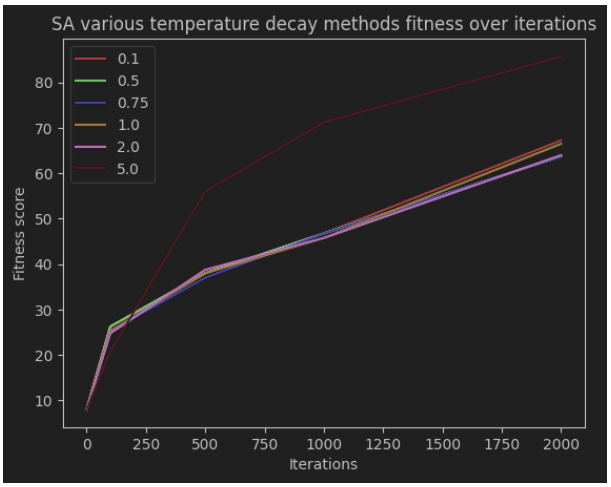


Fig 3. CCP SA various initial temperature over iterations

2.1.3 GA models

After chosen problem size 100, the GA population size is compared to find the impact to fitness. The population size from 10 to 800 are compared as shown in Figure 4. It shows more population size leads to higher fitness score.

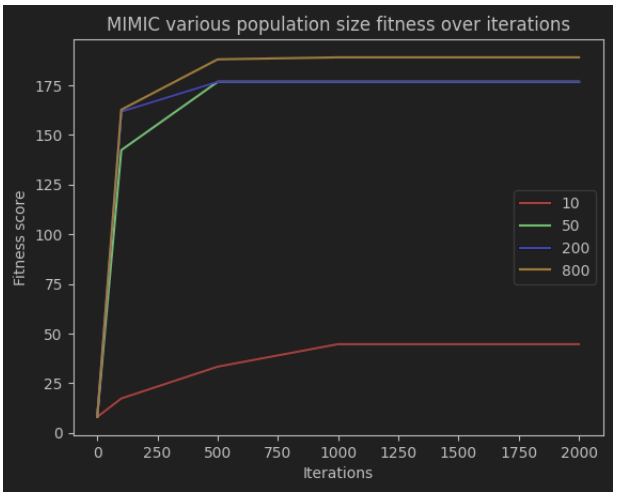


Fig 4. CPP GA various population size over iterations.

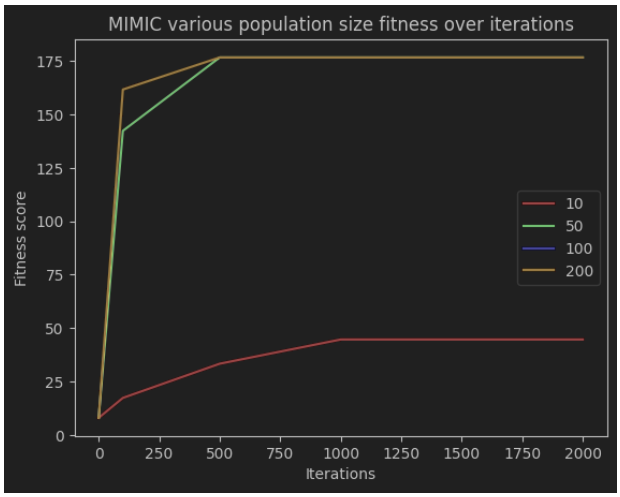


Fig 5. CPP MIMIC various population size over iterations

2.1.4 MIMIC model

After chosen problem size 100, the population size of MIMIC from 10 to 200 also compared to find the impact, as shown in Figure 5. It shows with population size increases MIMIC fitness score also increase. But lower population is already able to get best fitness, just needs more iterations.

2.1.5 Algorithms comparison

For CPP with problem size 100, the four algorithms average fitness score and average runtime are compared in Figure 6 and Figure 7. It shows GA gets the highest fitness, RHC and SA are much worse than GA and MIMIC. However, GA is much slower than all other algorithms. Compare with the problem size results in Figure 1, we can conclude that for CPP, with lower problem size (length of data), PHC and SA are better to get quick optimization results. But with higher problem size, GA and MIMIC has higher fitness score, but will leads to much higher run time.

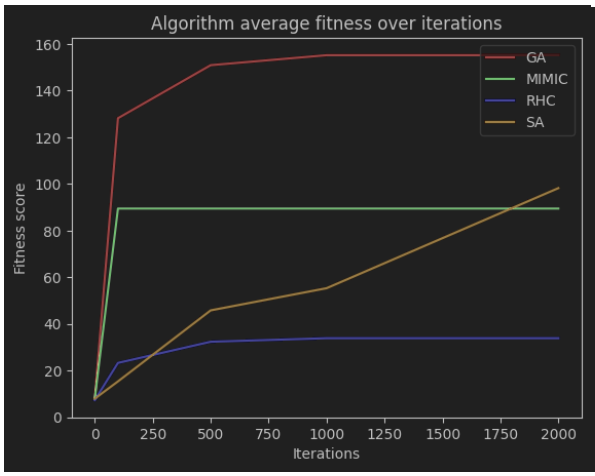


Fig 6. CPP Algorithm average fitness over iterations

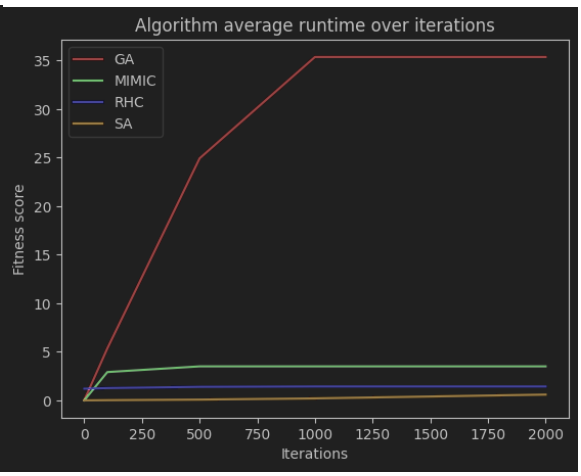
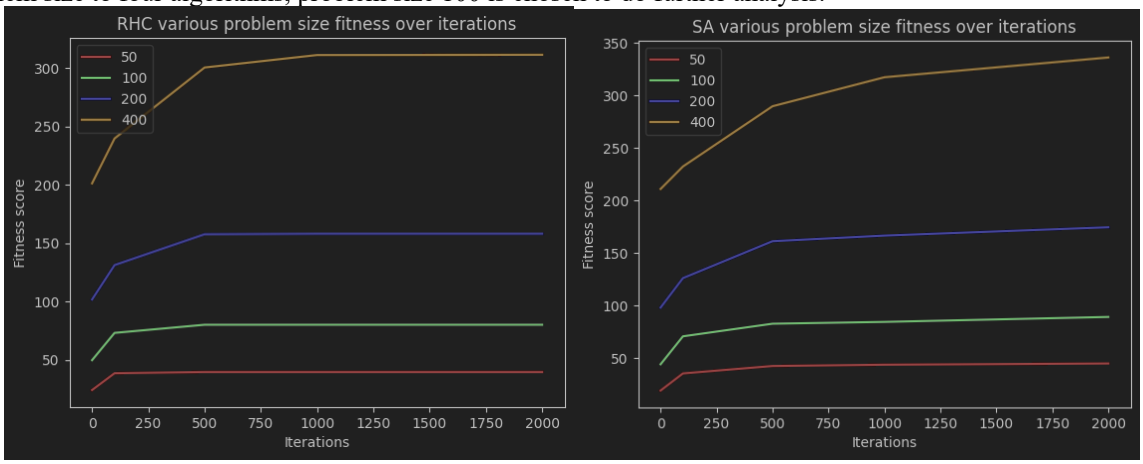


Fig 7. CPP Algorithm average runtime over iterations

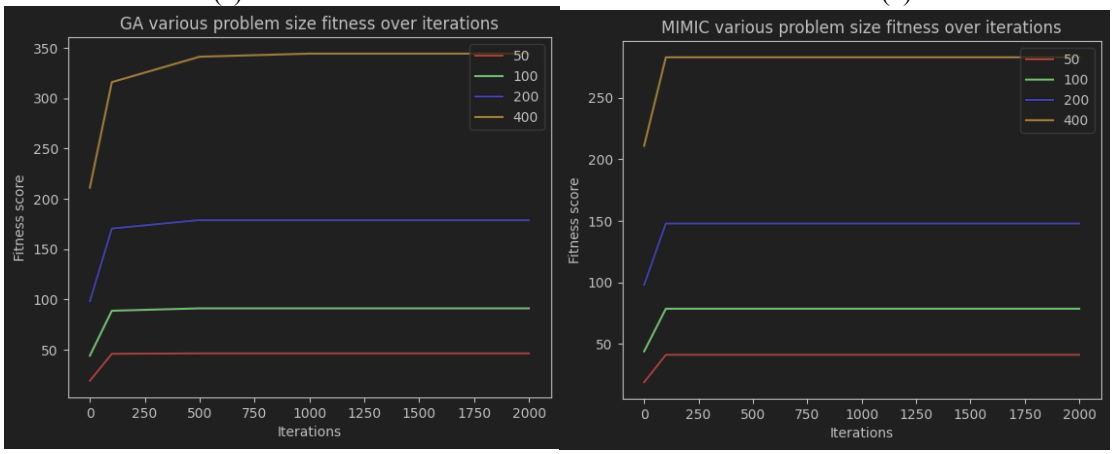
2.2 Flip Flop Problem (FFP)

The Flip Flop problem is to find the total number of consecutive bit alternations within a bit string. The highest fitness bit string consists entirely of alternate digits.

The problem size impact of all four algorithms in Flip Flop problem is shown in Figure 8. It shows with problem size increasing for FFP, all four algorithms fitness increase with problem size increasing. After comparison the impact of problem size to four algorithms, problem size 100 is chosen to do further analysis.



(a) (b)



(c) (d)

Figure 8. FFP four algorithm various problem size fitness comparison over iterations

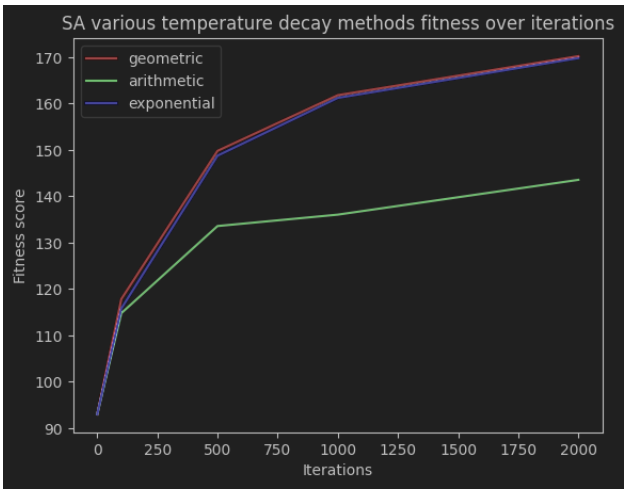


Fig 9. FFP SA temperature decay over iterations.

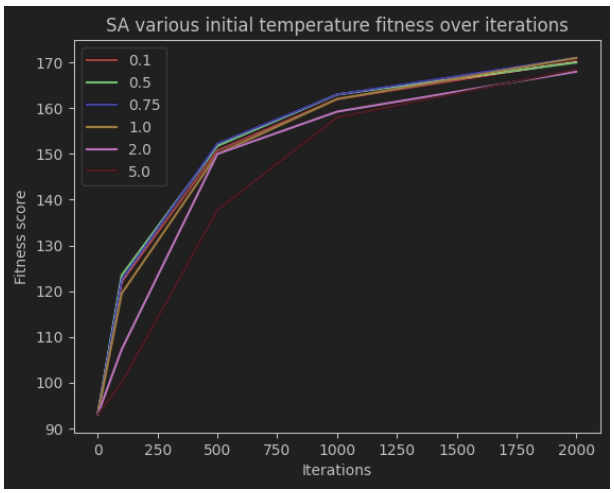


Fig 10. FFP SA various initial temperature over iterations

2.2.1 RHC models

As Figure 8(a) shows, the best average fitness is close to 60. It reached best fitness at around 1000 iterations for all 50 to 400 problem size. RHC fitness on larger problem size is significantly better than smaller problem size.

2.2.2 SA models

After chosen problem size 100, the compared 3 Temperature Decay methods are Exponential Decay, Arithmetical Decay and Geometrical Decay results are shown in Figure 9. It shows Geometric Decay fitness score performed better with lower iterations for FFP.

Then from Figure 10, various initial start temperature fitness comparison, it shows start temperature of 0.75-1 leads to higher fitness score for FFP.

2.2.3 GA models

After chosen problem size 100, the GA population size is compared to find the impact to fitness. The population size from 10 to 800 are compared as shown in Figure 11. It shows more population size leads to higher fitness score.

2.2.4 MIMIC model

After chosen problem size 100, the population size of MIMIC from 10 to 200 also compared to find the impact, as shown in Figure 12. It shows with population size increases MIMIC fitness score also increase.

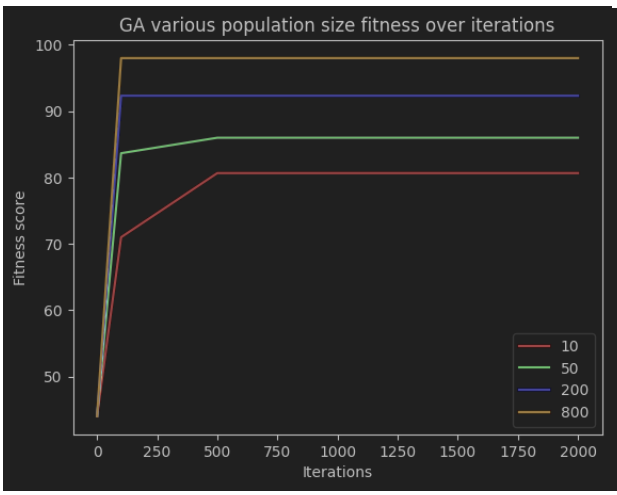


Fig 11. FFP GA various population size over iterations.

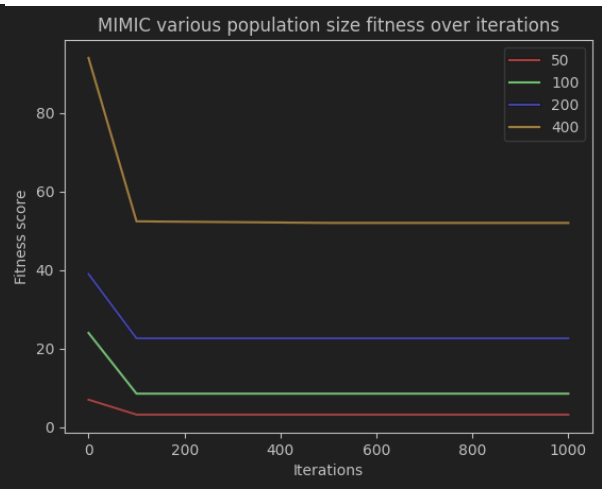


Fig 12. FFP MIMIC various population size over iterations

2.2.5 Algorithms comparison

For Continuous Peaks problem with problem size 100, the four algorithms average fitness score and average runtime are compared in Figure 13 and Figure 14. It shows GA gets the highest fitness, But SA is performed the second best.

However, GA is much slower than all other algorithms, and SA is much faster than the other 3 algorithms. Compare with the problem size results in Figure 8, we can conclude that for FFP, SA is the fast and better fitness algorithm to use.

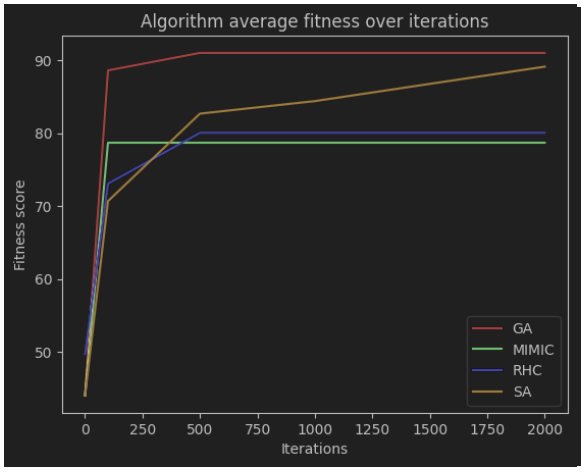


Fig 13. FFP algorithms average fitness over iterations

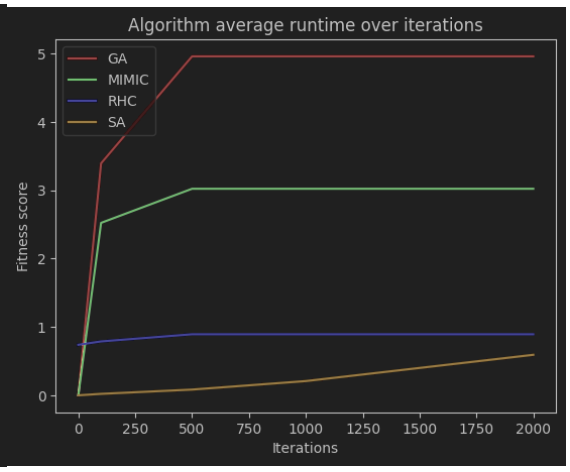
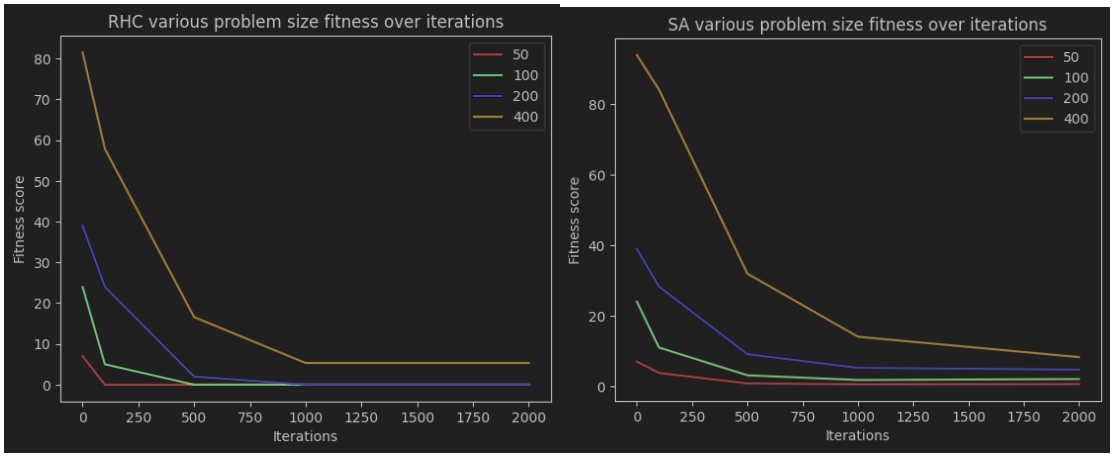


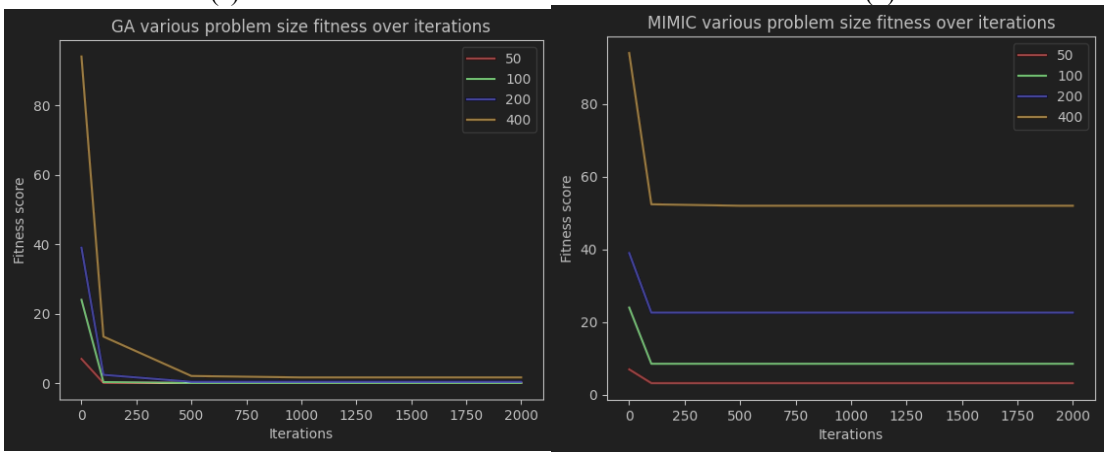
Fig 14. FFP algorithms average runtime over iterations

2.3 Max K Color Problem (MKCP)

The MKCP is to find out the minimum number of colors to color the blocks in the map with no same color on adjacent blocks. The less color used the better in MKCP, therefore, the optimization and fitness score is the lower the better.



(a) (b)



(c) (d)

Figure 15. MKCP three algorithms various problem size fitness comparison over iterations

The problem size impact of all four algorithms in MKCP is shown in Figure 15. It shows with problem size increasing all four algorithms fitness decrease with problem size increasing from 50 to 400. After comparison the impact of problem size to four algorithms, problem size 100 is chosen to do further analysis. 1000 iterations are already good enough for best fitness.

2.3.1 RHC models

As Figure 15(a) shows, it reached best fitness at around 1000 iterations for all 50 to 400 problem size.

2.3.2 SA models

After chosen problem size 100, the compared 3 Temperature Decay methods results are shown in Figure 16. It shows Geometric Decay fitness score performed better with lower iterations for MKCP.

Then from Figure 17, various initial start temperature fitness comparison, it shows start temperature of 1.0 leads to best fitness score for MKCP. Higher or lower temperature will lead to worse fitness.

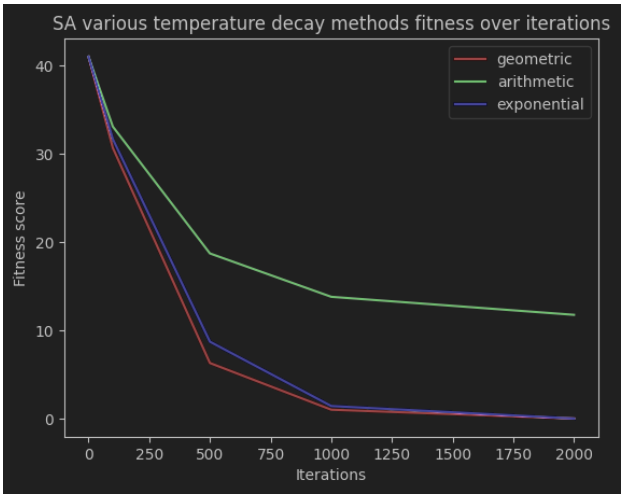


Fig 16. SA temperature decay fitness over iterations.

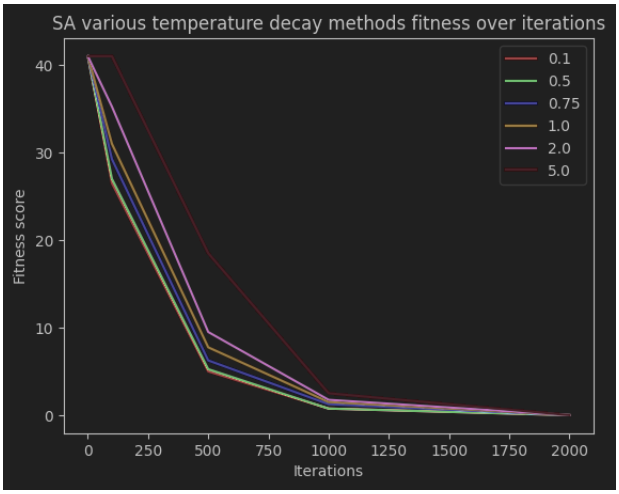


Fig 17. SA various initial temperature fitness over iterations

2.3.3 GA models

After chosen problem size 100, the GA population size is compared to find the impact to fitness. The population size from 10 to 400 are compared as shown in Figure 18. It shows more population size leads to better fitness score.

2.3.4 MIMIC model

After chosen problem size 100, the population size of MIMIC from 10 to 100 also compared to find the impact, as shown in Figure 19. It shows with population size increases MIMIC fitness score also getter better and reach to best fitness faster.

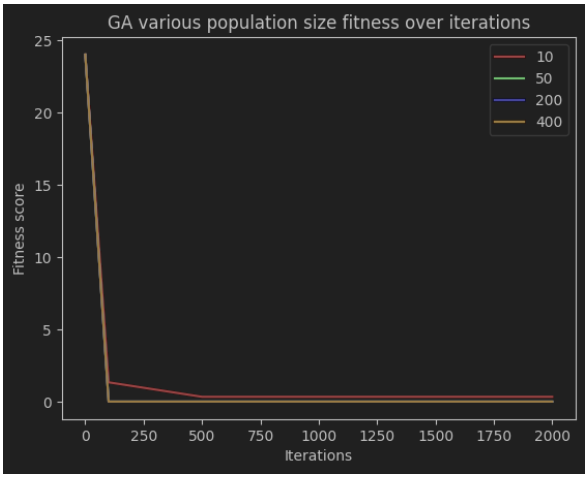


Fig 18. MKCP algorithms average fitness over iterations

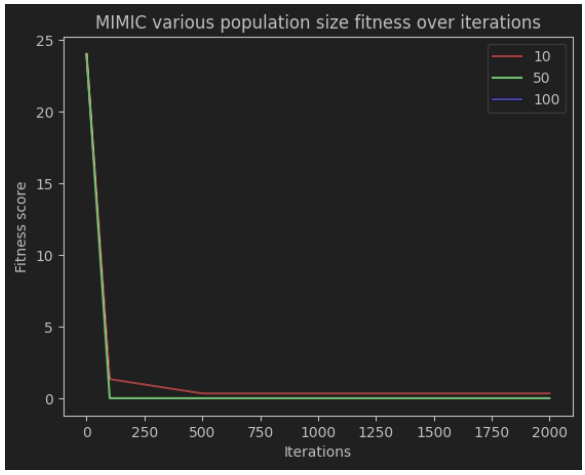


Fig 19. MKCP algorithms average runtime over iterations

2.4.5 Algorithms comparison

For MKCP with problem size 100, the four algorithms average fitness score and average runtime are compared in Figure 20 and Figure 21. It shows GA gets the best fitness, But RHC is performed the second best. However, GA and MIMIC are much slower than other algorithms, and RHC is much faster than GA and MIMIC. Compare with the problem size results in Figure 15, we can conclude that for MKCP, RHC is the fast and better fitness algorithm to use. But to get best Fitness with time trade off, GA has the best fitness.

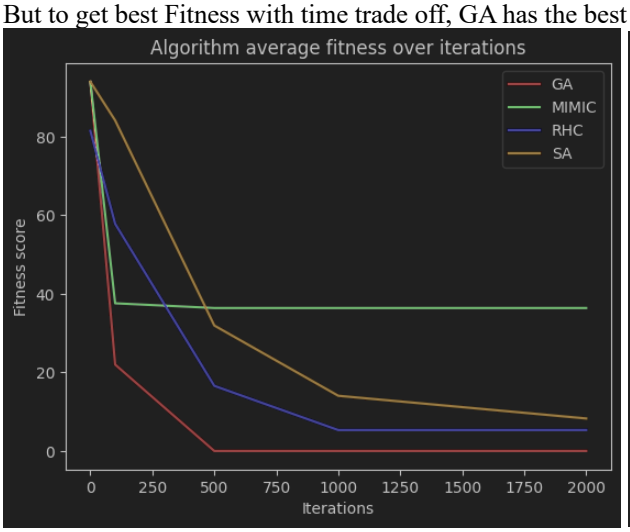


Fig 20. MKCP algorithms average fitness over iterations

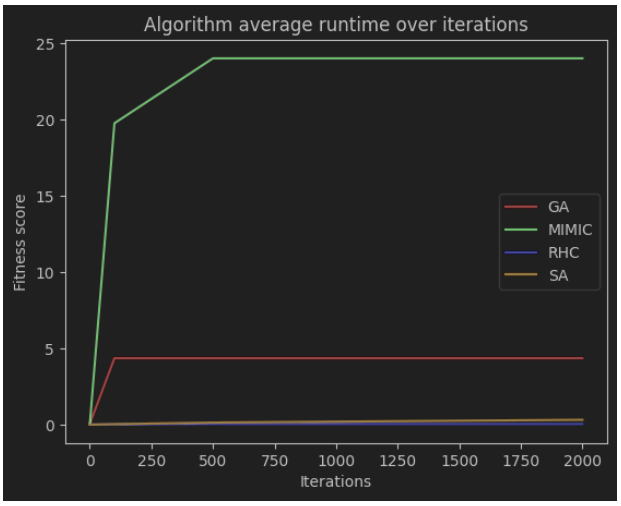


Fig 21. MKCP algorithms average runtime over iterations

2.4 RO algorithms comparison summary

The general comparison for the 4 RO algorithms is shown in Table 1.

Table 1. Neural Network optimization for 4 algorithms

Algorithm	Speed	Pros	Cons
RHC	Fast	Simple structure problem, sensitive to running time and cost	Not the best fitness
SA	Fast	Simple structure problem, sensitive to running time and cost	Not the best fitness
GA	Slowest	Structured problem, higher fitness demand, OK to trade fitness with time	Supper slow and costly with high population
MIMIC	Slow	Structured problem, higher fitness demand, OK to trade fitness with time	Supper slow and costly with high population

3. Neural Network weight optimization

RO can also be used in continuous problem like Neural Network (NN) weight optimization. In this part SA, GA, MIMIC and normal Gradient Descent (GD) are compared to find out how good RO can be in Neural Network training and compare the accuracy and F1 score to GD.

3.1 Test data

The data used is the COVID death dataset[3] that has 838,860 records with 61,539 deaths (7.336%). There are 1 million records and 19 useful columns in the dataset. The columns used are MEDICAL_UNIT, SEX, PATIENT_TYPE, DATE_DIED, INTUBED, PNEUMONIA, AGE, PREGNANT, DIABETES, COPD, ASTHMA, INMSUPR, HIPERTENSION, OTHER_DISEASE, CARDIOVASCULAR, OBESITY, RENAL_CHRONIC, TOBACCO, CLASIFFICATION_FINAL, ICU. Most of the columns are categorical data. Therefore, one hot encoding is used for most of the columns to generate features except AGE and DATE_DIED column. AGE is numerical feature itself; normalization might be needed for non-tree-based Machine Learning algorithms. DATE_DIED column is converted to target column as of death or not. After one hot encoding, the final features number increase from 19 to 66.

The Train test data split ratio is 80:20. After splitting, train data has 838,860 records with 61,539 deaths (7.336%). While test dat a has 209,715 records with 15,403 deaths (7.345%).

As in Project 1, the BP trained best Neural Network model used two layers of [25, 25] with "relu" activation function.

3.2 Basic RO algorithms comparison to GD

Firstly, it's general interest to briefly compare the model performance on 3 RO algorithm with GD. In this experiment, max_iteration is set as 50, leaning_rate set as 0.1, population set as 100, mutation_probability set as 0.1 to do general comparison.

The model accuracy for the four algorithms is shown in Table 1. The result shows all the three RO algorithms RHC, SA, GA performed extremely bad compared to GD, the normal back propagation method. However, GA performed better than GD with 50 iterations. But from the training time, we can see to get this result, GA training time is taking 40X higher than GD. Further experiments are needed to find out the if its iterations number limited the performance of GD. No wonder I've only seen papers using GA to optimize initial model weight for Neural Networks, but never use it to replace GD model training.

Table 1. Neural Network optimization for 4 algorithms with 50 iterations

Algorithm	Test Accuracy (%)	Train time (s)	Inference Time
GD	92.66	96	0.15
RHC	7.50	60	0.19
SA	7.50	77	0.19
GA	93.65	3815	0.17

3.2 GA population size impact

Since GA can optimize NN model, it would be interesting to find out the impact of GA population size to model training. The make it as fair comparison, all experiment are using maximum 200 iterations with early stopping. The NN model optimized by GA with population size 25 to 200 is shown in Table 2. It shows with the population size increases, GA can get better accuracy and F1 score. However, it will also greatly increase the training time. For GA with population size of 100, the test accuracy, F1 score and training time is lower than GA with population size of 50. This is not aligned with other results. This is probably due to early stopping. However, due to the intense deadline of this project, there is no time to rerun the experiments with early stopping turning off.

Table 2. GA Neural Network optimization for various population size

Population size	Train time (s)	Test Accuracy (%)	F1 score	Inference Time (s)
25	2797	93.71	0.367	0.18
50	17238	94.09	0.427	0.18
100	12375	93.90	0.404	0.18
200	59940	94.32	0.458	0.18

3.3 GA on more layers of NN

With GA capable of optimizing NN, a further question comes, that why people not using them to train NN and can GA be used in training more layers of NN and eventually Deep Learning models? To answer this question, a series of experiments with number of layers increases from 1 to 4 is compared. The result is shown in Table 3.

As the result shows, with the number of layers increases, the model accuracy only slightly increases. And the F1 score even getting worse. Due to the nature of COVID death prediction that there are way fewer deaths than survivors, F1 score would be a better metric for evaluation.

Therefore, for COVID death prediction problem, GA with a greater number of layers didn't really improve the model F1 score, but greatly increases the training time and inference time. And from the GD training time in Part 3.1, we can see GA will take 10X-100X more training time than GD to get slightly better accuracy. And with a greater number of layers and training time and inference time also greatly increases. It will not worth it to use GA to train complex NN and Deep Learning models with hundreds of layers.

Table 3. GA Neural Network optimization for various layers

Layer size	Train time (s)	Test Accuracy (%)	F1 score	Inference Time (s)
[25]	10180	92.93	0.457	0.12
[25, 25]	12376	93.90	0.404	0.17
[25, 25, 25]	19097	93.63	0.406	0.24
[25, 25, 25, 25]	24793	93.50	0.331	0.31

3.4 Summary

With all the comparison above, we can conclude, RHC and SA is not good fit to optimize NN weight. GA can optimize NN, but will take way more time than normal Gradient Decent. However, due to GA takes 10X to 100X more time to train, it will be not feasible to use GA to optimize Deep Learning NN with dozens and hundreds of layers.

4. Conclusion

In this analysis report, we first compared the fitness and run time performance for four RO algorithms (RHC, SA, GA and MIMIC) on 3 common problems (Continuous Peaks Problem, Flip Flop Problem and Traveling Salesman Problem).

Results shows in CPP, GA gets the highest fitness, but much slower than other algorithms. In FFP, SA is the fast and the second better fitness. In MKCP, RHC has the second-best fitness, and it's the fastest.

Then we compared 3 RO algorithms (SA, GA and MIMIC) on Neural Network optimization with normal Gradient Decent. Result shows only GA can optimize NN weight and perform better than GD for small size of NN model. Due to GA is taking way too much time, it's not worth it to use it on NN with high number of layers.

References

1. *Hayes, Genevieve. mlrose: Machine learning, randomized optimization and search package for python. (2019): 35.*
2. *Rollings, A. (2020). mlrose: Machine Learning, Randomized Optimization and SEarch package for Python, hiive extended remix. https://github.com/hiive/mlrose.*
3. *Kaggle COVID-19 Dataset: https://www.kaggle.com/datasets/meirnizri/covid19-dataset?resource=download*