7642 Assignment 2 Report

Zebing Li

Georgia Institute of Technology
Atlanta, Georgia
zli738@gatech.edu

I. Introduction

Randomized optimization is a widely used optimization technique that can find out optimal solutions to complex problems effectively. It involves generating random sampling to get candidate solutions in order to achieve global optima. Despite their simplicity, optimization optimization can be especially effective. And therefore, there are a variety of applications, like hyper-parameter tuning, or solving combinatorial optimization problems. There are various types of optimization approaches, such as random hill climbing, simulated annealing, genetic algorithms, and MIMIC. They have different features, preferences and biases, and hence they might behave very differently under different circumstances. Given the variety of the techniques and problems, it is crucial to understand the advantages and disadvantages of the algorithms, and choose one or more suitable techniques based on the feature of the optimization problem in order to solve the problem more effectively and efficiently.

In this paper, I explored four randomized optimization techniques, random hill climbing, simulated annealing, genetic algorithms, and MIMIC. In order to compare their performances, I chose two different discrete optimization problems to highlight each algorithm's advantages or disadvantages. And the performances are evaluated from various aspects like fitness, running time, number of iterations, e.t.c. After this, I also applied the first three algorithms expect MIMIC to a neural network problem, and use the algorithms to optimize the weights and compare their performances with back propagation.

The remaining paper is structured as following: first I will introduce the two optimization problems, and the neural network dataset. And I discuss the reasons why chose the problems, and hypothesises of each problem. Then I will share the results with figures and tables for the two discrete problems and neural network. And I will analyze the results and discuss the findings.

II. Problems

A. Four Peaks

The first problem I chose is Four Peaks problem. It involves a binary string, and the fitness function counts the number of consecutive leading 0s and number of consecutive trailing 1s and returns the maximum. And if both number exceeds a threshold, an extra reword is added.

Here's the fitness function:

$$f(S) = max(N_0, N_1) + R; if N_0 > T and N_1 > T$$
$$= max(N_0, N_1); otherwise$$

Where N_0 and N_1 are number of 0s and 1s; R is predefined reward. And T is predefined threshold.

I think this is an interesting problem to look at. First, it has a very simple and straightforward fitness function and the problem is very easy to understand. And it can be easily applied to different algorithms without adjustment. Therefore, it is a good algorithm testing problem. Second, it has multiple "peaks" as described its name. There are small peaks with either a lot of 0s or a lot of 1s, and larger peaks when the reward is applied. Hence, the landscape is complex and there might be multiple local optima. It is interesting to see whether different algorithms can achieve the global optima. Also, looking at the time complexity, it takes linear time to scan through the string and evaluate the fitness. However, the choice of algorithms would affect the total number of evaluations and acquire different amounts of time.

Given the feature of the 4 peaks problem, my hypothesis is simulated annealing would perform better because of its capability of escaping local optima. And because the problem is not very complex, starting with a single solution and continue to refine the single solution is more efficient.

B. Max K Color

The second problem is max K color. It is a graph problem. Given a graph, the goal is to color the vertices of the graph using at most K colors, so that the number of edges with different color on its endpoints is maximized.

Here's the Problem Definition:

Given:

- A graph G=(V,E)G = (V, E)G=(V,E) with vertices V and edges E.
- An integer k representing the maximum number of colors.

Objective:

- Maximize the number of edges with vertices of different colors.

I chose the problem because it is a NP-Hard problem, meaning there's no known polynomial time algorithm to solve this problem, and it takes potentially exponential number of

evaluations to find out an optima. Therefore, it is a more complex problem compared with 4 peaks. Therefore, the algorithms might perform differently from problem 1. And it might highlight algorithms that works better in more complex space. Also, as the sizes of the graph grows, the complexity grows exponentially. So varying the problem size will make more significant effects and demonstrate the differences of the algorithms.

Given the complexity of the max K color problem, my hypothesis that Genetic algorithm and MIMIC would perform better than others. Because they both evaluate a population of solutions and explore the search space more thoroughly.

C. Neural Network

The problem I chose for the neural network is Mushroom classification. It is one of the datasets I used for assignment 1. It is a binary classification problem, giving different features of a mushroom and to predict whether it is edible or not. There are 9 features describing aspects of caps, gills, stem, etc. And the target class is edible (0) or not edible (1). Among the features, cap shape, gill color and stem color are categorical features, others are numerical. There are 5100 rows of data, and I chose 80 percent as training data and 20 percent as testing data. I used cross validation in training by equally splitting the training data to 5 pieces. The data is pre-cleaned and it is not very noisy. I applied standard scaling on the non-category features, since they are of very different scales.

This dataset has some interesting characteristics. First, the dataset is relatively imbalanced. The target class is 62% edible and 38% inedible. Some features are imbalanced too, such as gill color and stem color. Second, the features have very different scales and distributions differ a lot among each other. Third, it includes some irrelevant features in the dataset, like cap-diameter which has very small correlations with the target.

From assignment 1, neural network was able to model the complexity of the dataset pretty well. It was able to achieve around 0.95 accuracy for validation and testing. Also, it takes significant amount of training time as neural network has complex structure and back propagation can take long to train and tune the weights. Also, neural network performed better for label = 0 as there were more data to train. And the performance is less optimal for label = 1. It would be interesting to compare the back propagation results with random optimization results.

Since weights of neural networks are continuous, and the space is very complex, my hypothesis is that GA can perform better than other random optimization algorithms in terms of accuracy. But it would take significant amount of time.

III. Methods

For implementation, I used python library mlrose_hiive's implementation of random optimization algorithms and the discrete optimization problems. For neural networks, I used scikit learn implementation MLPClassifier for back propogation, and used mlrose_hiive for other random optimization algorithms.

For discrete problems, I first fix the problem size, and compare best fitness, time to reach best fitness, iterations to reach best fitness and total running time. Then I plotted fitness vs number of iterations, function evaluations vs iterations, function evaluation vs wall clock time to visualize the differences. After this, I varied the problem sizes and looked into differences on fitness/wall clock time/function evaluation/ iteration and time to reach best fitness vs problem sizes.

For the neural network problem, learning curves with respect to training sizes and training time are included to show the training progress and it also gives some hints about over-fitting. The main metric used to measure the model performance is F1 score, as it is a useful metrics on binary classification and is very useful on imbalanced data as the dataset here. I also used confusion matrix as it provides a detailed breakdown of each group.

IV. Results

A. Four Peaks

1) Fix Problem Size: First we fix the string length = 8, and compare differences on other aspects. I also fixed the max iterations = 100 as I noticed all four algorithms converge well within 100 iterations. And I set max_attempts=500 as more attempts leads to more exploitation to find a better neighbor. But too many attempts would take long time and might reduce efficiency. Since this is a relatively small problem, I set it to be 500 which turns out to be adequate.

For simulated annealing, I tuned temperature and decay function. For temperature, I tried [0.01, 0.5, 1.5, 2]. It showed that low temperature as 0.01 made it slow and required more iterations to reach the optima, while higher temperature like 2 make it reach the global optima and then moved away. This is because a too high temperature tend to make it move away to explore other points. Therefore I chose 0.5 as the temperature. For decay function, I tried GeomDecay, ArithDecay, and ExpDecay. ArithDecay decays slowly, and expDecay decays too fast. They both lead to a local optima. I think this is because ArithDecay does not do enough exploration, and ExpDecay moved away from global because heat is still high. Therefore, I chose GeomDecay which cools fast initially then gets slower.

For Genetic Algorithm, I tuned population sizes and mutation rate. I experimented population size = [10, 50, 200]. When it is 10, the GA algorithm doesn't reach global optima. This is because the potential solution population is too small so it doesn't converge. For population size =20, GA can converge to global optima but takes significantly longer time. Therefore, population size is set to 50 to make sure convergence while less time consuming. For mutation rate, I tried [0.1,0.2,0.5,0.9]. It seems that all of these rates can lead to global optima in similar time and iterations. I think this might be because the problem is simple and population size is large enough so it doesn't need too much mutation to converge. However, larger mutation rate causes slightly longer running time. I set mutation rate = 0.2 to make sure there are some random changes.

For random hill climbing, I tuned restarts. I tried restarts = [1,2,3,4,5]. Since each restarts run a separate 100 iterations, it turns out not all restarts can lead to global optima. There are always some restarts iterations end up with a local optima. And naturally, one more restart increase one more round of time. But more restarts will have larger chances to have one starting point reaches global optima. I chose 3 restarts at end.

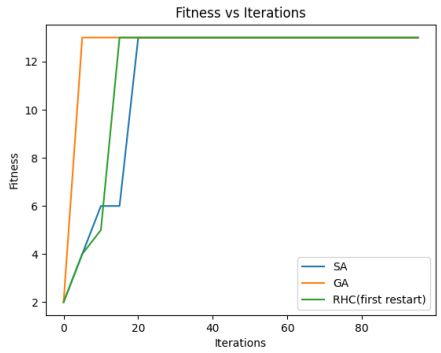

Fig. 1. Fitness vs Iterations

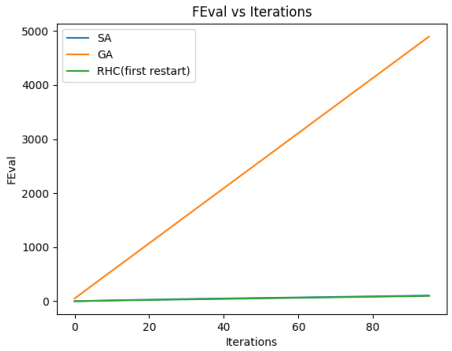

Fig. 2. Feval vs Iterations

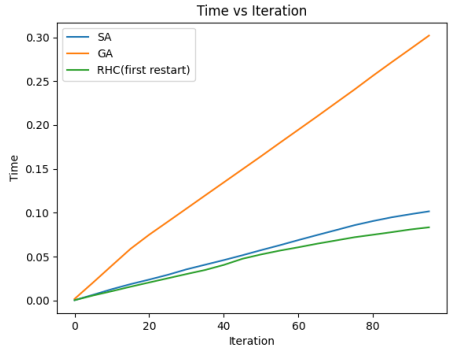

Fig. 3. Time vs Iterations

Now we look at the plots. Notice that since RHC has restart rounds of iterations, it has more rounds of iterations than other two algos. In plotting, I only plotted the first round of iterations

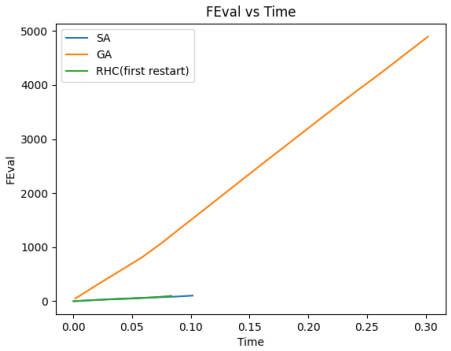

Fig. 4. FEval vs Time

to compare with other algorithms. From the Fig 1. Fitness vs Iterations, we can see that all three algorithms reaches same final fitness. GA requires least iterations, RHC and SA use similar iterations to reach best fitness. Looking at Fig 2, Feval vs iterations, it's clear that GA requires much more number of fitness function evaluation that other two algorithms. The slope of feval vs iteration is roughly 50 which is the population size, while the slope is roughly 1 for RHC and SA. This is because for each iteration, GA evaluate a population of solutions and has to evaluate the fitness for each solution. Therefore its fevals is linearly correlated with population size. And SA and RHC works with a single solution at a time. This is why GA needs less iterations to converge, but each iteration takes more fevals and hence much more time. This is shown in fig 3 time vs Iterations as well. From fig 4, we can see each iteration for SA takes slightly more time than RHC. This is because RHC simply moves to its neighbor if it is better while SA involves randomness and decays the temperature. From Fig 4 Feval vs Time, we can see GA feval per time is higher than other two algorithms. This is again because it involves a large amount of intensive evaluation. From the x axis, we can also see that GA takes more time. Although RHC has least time for one single iteration, it restarts so total time would be larger than SA.

Table 1 is a summary of the performances. For fitness, all three algorithms get same best fitness. However RHC gets a local optima for final fitness because the last restart falls into a local optima range. In terms of total time, SA takes least time while GA takes most of time. Similarly, SA needs least number of feval, while GA requires most. Looking at the time and iterations to reach best fitness, we can see GA wins as it takes least number of iteration and time to reach best fitness. While SA takes most iterations to get to best fitness. In conclusion, SA wins in total time, but as a trade off it requires more iterations. GA requires least amount of iterations, however as a trade off, it requires more feval and more time. RHC is in the middle in terms of time and fevals, but it might be stuck in local optima in some of its restarts. For this size, clearly SA is the best performer as it gets global optima very efficiently and effectively.

	SA	GA	RHC
Best Fitness	13	13	13
Final Fitness	13	13	8
Total Time	0.10	0.30	0.20
Total Feval	104	4897	394
Time to reach best fitness	0.02	0.01	0.02
Iteration to reach best fitness	20	3	13

TABLE I
FOUR PEAKS

2) Vary Problem Size: In this step, I tried string lengths = [10, 20, 40, 80], and to observe the performances of the algorithms. As the problem size is larger, I increased the max iterations to 3000 to make sure the algorithms have enough iterations to convergence. Similarly, I increased the max attempts to 1000 to check for more neighbors.

For SA, I now set temperature = 2.5 to allow it have more exploration as the problem is more complex and might have more local optimas. I also increased mutation rate for GA to 0.5 and restart for RHC to 5 for same reason. The population size for GA is still 50 because the problem is not too complex and I think 50 is enough to Handel it.

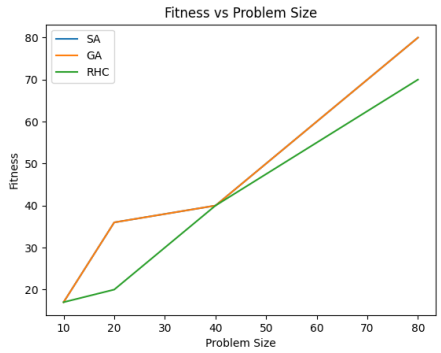

Fig. 5. Fitness vs Size

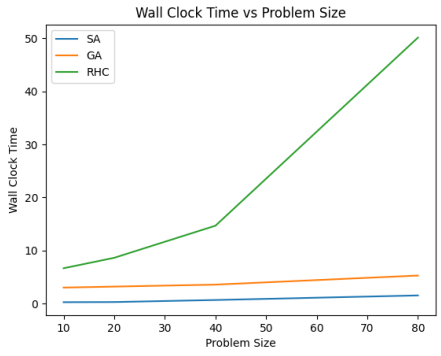

Fig. 6. Time vs Size

The fig 5 Fitness vs problem size plots final fitness vs problem size. We can see SA and GA agree on best fitness for all sizes, while RHC sometimes get less optimal final fitness. But this just shows that the last restart goes to a local optima,

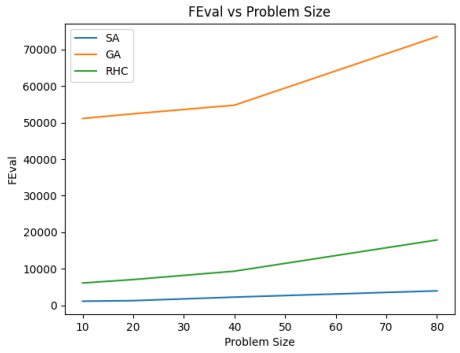

Fig. 7. Feval vs Size

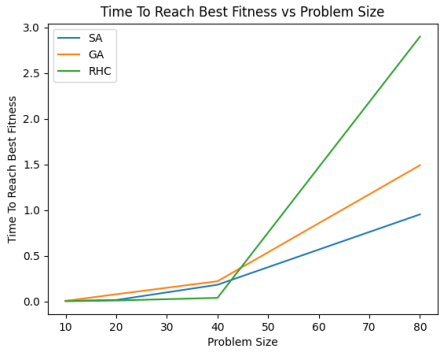

Fig. 8. Time to reach best fitness vs Size

actually some restarts achieves global optima. From fig 6Wall clock time vs problem size, we can see now RHC takes most time as it now has more restarts and each restart has more iterations. GA is second and SA takes least amount of time. This is because GA requires more function evaluations than SA. Looking at fig 7 Fevals vs Size, we can see still GA has much more fevals than other two algorithms due to its property to evaluate a basket of candidates. Followd by RHC as it has more restarts, followed by SA. Looking at Fig 8 iterations to reach best fitness, GA takes much less iterations same as

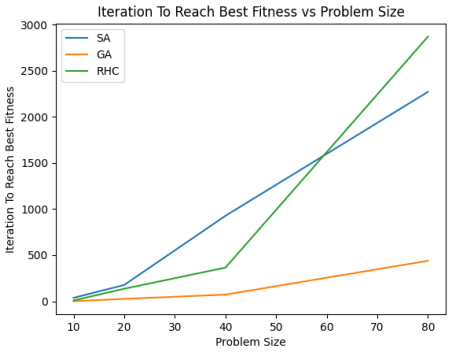

Fig. 9. Iteration to reach best fitness vs Size

the earlier analysis we did before. And SA starts with most iterations when problem size is small, but got exceeded by RHC as problem size grows. From fig 9. time to reach best fitness, we can see SA continues to perform very well in all problem sizes, while GA starts to show advantage as problem size grows, and in the meanwhile RHC performs better at smaller problem size.

In conclusion, SA is the best performer as it converges to global optima very efficiently for all problem sizes. This is because it refines a single candidate solution and allow it to make gradual improvements. This might be essential for the the problem to fine tune a possible solution. Also avoid local optima by randomly accepting worse solutions. Trade off is it might take more iterations to achieve best fitness than GA as it works with a single solution. GA can also converge to global optima but it takes more time as it works with a population of solutions. But this also helps it to converge in fewer iterations and its advantage starts to show up as problem size grows, in which case starts with a population of candidates would be more efficient. And RHC works better in smaller sizes. When size become larger, it takes more iterations and longer time as it always reach the local optima before it randomly starts again. So it is more easily trapped by a local optima when there are more deceptive local heals.

B. Max K Color

1) Fix Problem Size: First I fixed the problem size with 5 nodes and 8 edges. I used same max iterations and max attempts as Four peaks. max iterations = 100 as all four algorithms converge well within 100 iterations. max_attempts=500 which turns out to be adequate.

For simulated annealing, I tuned temperature and decay function. For temperature, I tried [0.01, 0.5, 1]. Same as four peaks, 0.01 made it slow and required more iterations to reach the optima, while higher temperature like 1 make it reach the global optima and then moved away. Therefore I chose 0.5 as the temperature. For decay function, I continued to use GeomDecay.

For Genetic Algorithm, I tuned population sizes and mutation rate. I experimented population size = [10, 50, 200]. When it is 10, GA algorithm can already reach global optima. I think this is because the problem size is very small and it doesn't require more population sizes. And larger population sizes can all converge well but they take longer time. So I used population size=10. For mutation rate, I tried [0.1,0.2,0.5,0.9]. It seems that all of these rates can lead to global optima in similar time and iterations as the problem size is small. I set mutation rate = 0.2 to make sure there are some random changes.

For random hill climbing, I tuned restarts. I tried restarts = [1,5,8]. For restarts = 1 and 5, it doesn't converge to a global optima therefore it needs more restarts. Therefore I set restart = 8 and it reached global optima.

For MIMIC, I used same population size = 10 as GA, and keep percent list = 0.2 as GA's mutation rate. As they can converge to global optima. I tried with larger population sizes and they are more time consuming. For keep percentage, I also tried with 0.5 and 0.8. But 0.8 starts to not converge to global optima because it has less exploration.

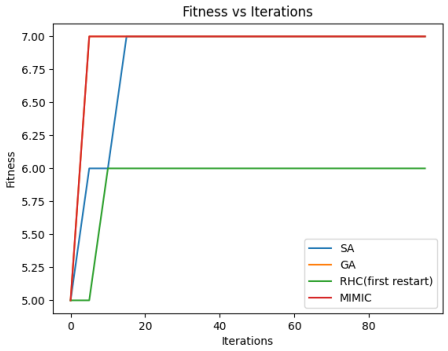

Fig. 10. iteration vs fitness

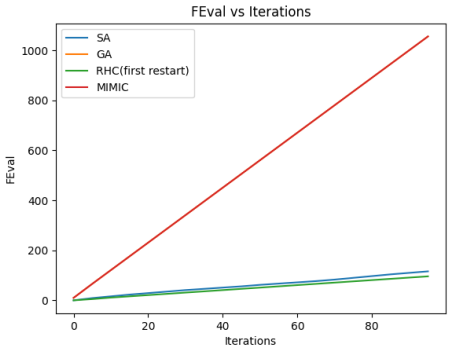

Fig. 11. feval vs iterations

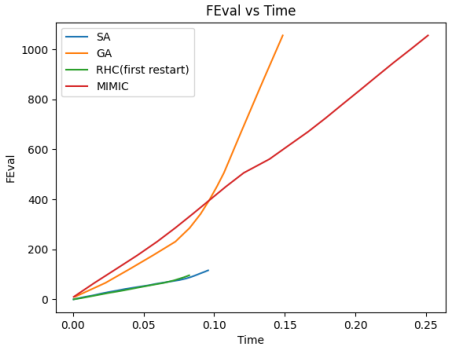

Fig. 12. feval vs time

Now we look at the plots. From Fig 10. and Fig 11, Fitness/Feval vs Iterations, we can see that Mimic and GA almost overlap with each other. They both reach max fitness in one iteration and both have large Feval per iteration. As they both handle a sample of candidates and has to evaluate them per iteration. From fig 13 and Fig 14, we can see MIMIC takes more time as iteration grows, and GA can do more fevals

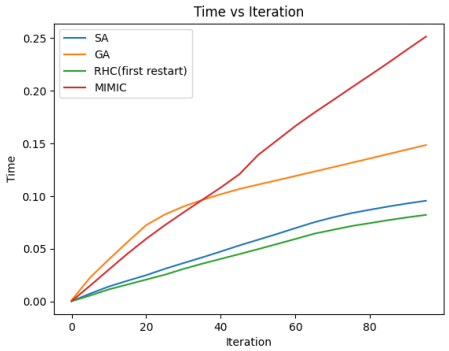

Fig. 13. iterations vs time

	SA	**GA**	**RHC**	MIMIC
Best Fitness	7	7	7	7
Final Fitness	7	7	7	7
Total Time	0.096	0.148	0.536	0.251
Total Feval	116	1056	864	1056
Time to reach best fitness	0.020	0.010	0.087	0.005
Iteration to reach best fitness	14	1	96	1

TABLE II
MAX K COLOR

per time, because it estimates the joint porbability distribution and mutual information which can be time consuming. From the table, we can see in terms of total time, SA is still more efficient but the differences is much smaller compared with 4 Peaks. This is because max K color is more complex space and GA/MIMIC effectively explore large search space due to population based approach. Also, we can see GA and MIMIC both uses 1 iteration to converge. RHC now is less efficiency from time and iterations, because it now needs more restarts to converge to global optima.

2) Vary Problem Size: In this step, I tried graphs with nodes = [10, 20, 40, 80], and edges = [10, 20, 40, 80].

For SA, I now set temperature = 1.5 to allow it have more exploration as the problem is more complex and might have more local optimas. Restart for RHC is set to 5. The population size for GA and MIMIC is still 50, and mutation rate/keep percent list are set to 0.5 to compare these two algorithms.

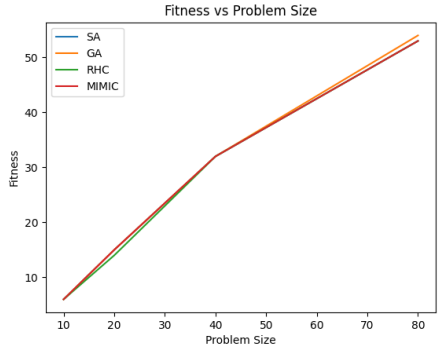

Fig. 14. fitness vs size

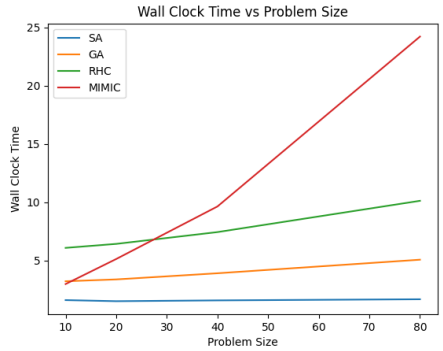

Fig. 15. time vs size

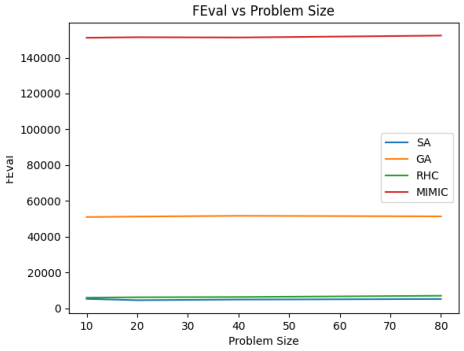

Fig. 16. feval vs size

First looking at the fitness plot Fig 14, we can see GA converges to maximum fitness all the time. MIMIC doesn't converge to max at largest size. I tried with a larger population size, MIMIC can converge but it requires more time. SA/RHC doesn't converge to max even for smaller sizes. From Fig 15 wall clock time, as size increases, MIMIC's time grows fast. GA's slope is relatively flat and it doesn't take much time to converge. RHC takes more time too and SA still takes least amount of time. From Fig 16 FEVAL, GA and MIMIC overlap at high value compared to SA and RHC. In terms of time and

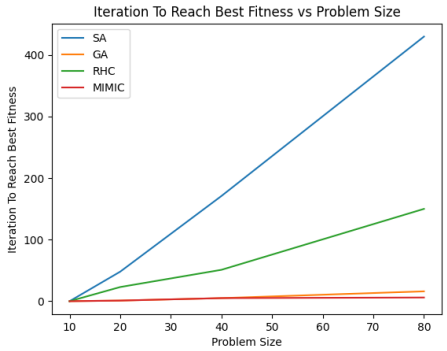

Fig. 17. Iterations to reach best fitness vs Size

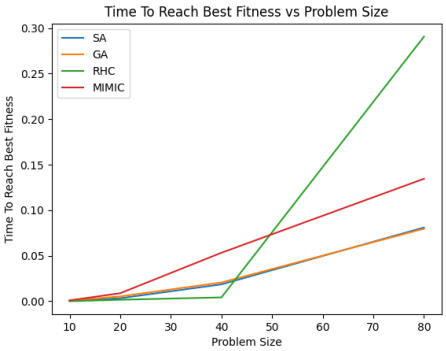

Fig. 18. Time to reach best fitness vs Size

iterations to converge from fig 17 and fig 18, both MIMIC and GA takes less iterations and time, but MIMIC's time per iteration is larger at large problem size.

In conclusion, this problem highlights the advantages of GA as it converges to global optima efficiently. As the graph becomes larger, the complexity grows exponentially, when GA can more effectively handle due to the population based approach and it maintains diversity which avoids local optima. MIMIC also performs well as it's also population based. But the nature that it calculates the joint probability makes it more computationally expensive and more time consuming. But its advantage is that it can explore more effectively using probabilistic sampling and model dependencies better. So if the graph getts even larger and more complex, MIMIC might outperform GA. SA doesn't work well in larger sizes of graph because it keeps refines a single solution and when the space is very large, it can't scan the whole space thoroughly and effectively.

C. Neural Networks

In assignment 1, I used hidden layer size = 40, and activation= logistic. But mlrose_hiive doesn't support logistic. To make the hyper parameters consistent, I use hidden layer size = 40 and activate = relu for all algorithms.

As neural network weights are pretty complex and they are continuous, I tuned the parameters of the algorithms to increase their complexity. I increased max iterations to 1000 because more iterations might be required to converge to a better accuracy. For GA, I used population size = 250 and mutation prob = 0.5 to have enough population to explore. For RHC, I set max iterations = 1000 and restart 3 times.

	Back Prop	SA	GA	RHC
Training Time	1.565	18.712	60.833	16.600
Accuracy	0.98	0.78	0.79	0.81
F1 Score	0.97	0.67	0.67	0.70

Looking at Table 3. For training time, back propagation is way faster than all randomized optimizations. Also, from the predicting scores, back prop is also much better than others. Looking at confusion matrix (Fig 19 - 22), clearly randomized optimization mispredict both labels much more than back prop.

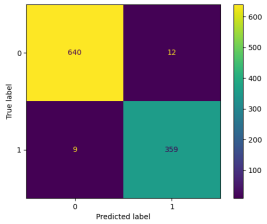

Fig. 19. Back Prop Confusion Matrix

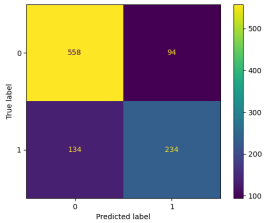

Fig. 20. SA Confusion Matrix

The reason is back prop uses gradient descent to systematically adjust the weights to minimize the error. Therefore it provides a clear direction for updating weights. And by propagating errors backward, the weights are updated in a very efficient way to converge to a local or global optima very quickly. However, for randomized optimization, the weights are updates randomly and it lacks for direction to update, which makes it longer to converge or even leads to unsatisfactory solution. Also, due to large scale and continouity, it becomes impractical due to immemse number of random trials. Comparing SA, GA and RHC, the performance doesn't differ much. GA requires

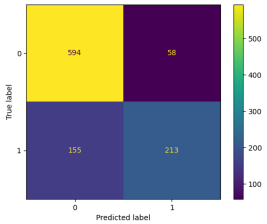

Fig. 21. GA Confusion Matrix

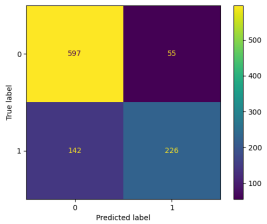

Fig. 22. RHC Confusion Matrix

most amount of time because it explore whole space and evaluate more times due to the population approach. RHC gets a higher accuracy because it continues to hill climbing at each restart and it is a clear guidance. SA doesn't take much time as it is not very time consuming but it also doesn't achieve very good accuracy.

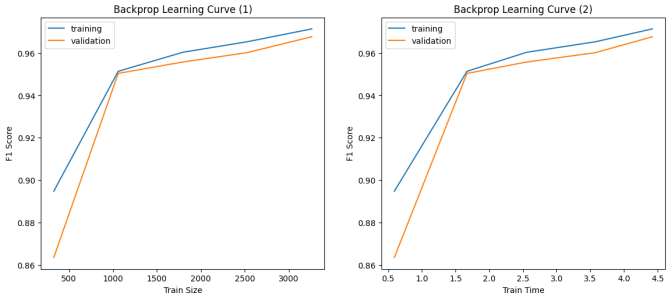

Fig. 23. Back Prop Learning curve

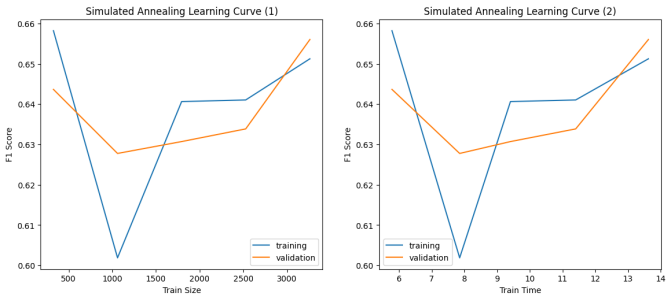

Fig. 24. SA Learning curve

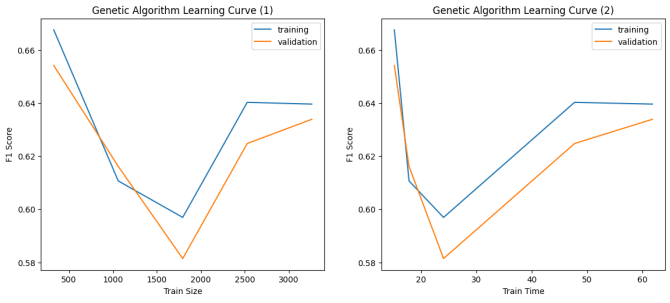

Fig. 25. GA Learning curve

Looking at the learning curves (Fig 23-Fig 26), I use F1 score as metric because it considers accuracy for unbalanced dataset. we can see for back propogation both training score and validation score continues to improve and stabilize at a high level at around 0.97. This is because it continues to get guidance from error function. So it can update the weights with a clear direction to minimize error and improve accuracy. However, looking at three randomized optimization algos, they all first have some decrease in training and validation scores. This is because at first, they have high randomness, like high temperature in SA in the begining, so they might

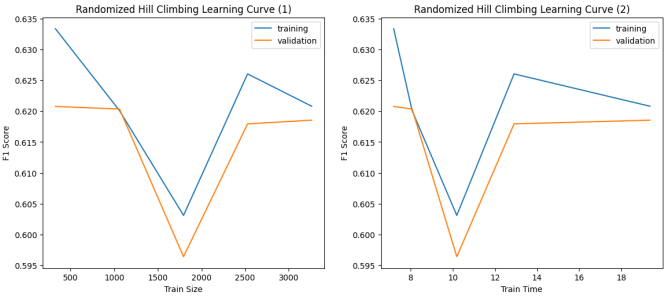

Fig. 26. RHC Learning curve

move to a less satisfactory neighbor. As more training goes, their performances start to get better. For SA, its temperature cools down, so it gets more exploitation towards better score. For GA, it takes longer time to decrease and then gradually move to a better population by selection and cross over. For RHC, it might reach different local optima as it restarts. But as it restarts more, it can reach to a better optima.

In conclusion, for neural networks, back propogation outperms all randomized algorithms. The main reason is it leverages gradient of loss function to make efficient and guided updates towards the weights. So it leads to a faster and better convergence to an optimal. However, randomized optimization lacks informative and systematice guidance to improve accuracy, and result in a much slower and less effective training. And due to the immense number of randomness in the weights, it is impractical for randomized optimization to adapt to large neural networks.

V. Conclusion

I investigated different randomized optimization algorithms and apply them on two discrete optimization problems and neural networks. Four peaks problem highlighted the advantages of simulated annealing as it converges to global optima most efficiently. Genetic algorithm is more optimal for Max K color problem especially when the graph size is large. MIMIC algorithm is also a good choice for large max K color problem but it takes more time to converge compared with GA. Overall, SA is more suitable for smaller and simpler spaces as it focus on improving and refining a single solution which can be very efficient in some cases. GA and MIMIC perform better when the space is more complex as they work with candidate populations and they can explore the space more thoroughly. As a trade-off, they can be more computationally expensive. MIMIC is more time consuming due to estimation of the probability densities. For neural networks, back propagation works much better than randomized optimization algorithms. This is because randomized optimization does not have a systematic improvement process from gradients and it relies on random trails, hence it can be very slow or hard to converge to an optima given the complexity of the neural networks.