CS 7641 Assignment 2: Randomized Optimization

Ali Alrasheed
College of Computing
OMSCS

***Abstract*—This paper compares the performance of different alternative algorithms for Gradient Descent in optimizing the weights of Neural Networks. In particular, Random Hill Climbing (RHC), Simulated Annealing (SA), and Genetic Algorithm (GA) are shown to be able to produce a similar performance to Gradient Descent, although they take significantly more iterations. The paper also discusses the performance of Random Hill Climbing (RHC), Simulated Annealing (SA), Genetic Algorithm (GA), and MIMIC in three different optimization problems. The Continuous peaks optimization problem demonstrates the strength of SA compared to other algorithms. In addition, Knapsack problems aim to highlight the superiority of using GA. Finally, Flip Flop is used to highlight the benefits of using MIMIC.**

***Index Terms*—ML, ANN, SA, GA, MIMIC, GD, Flip Flop, Continuous Peaks, Knapsack, Heart Disease Dataset.**

I. Introduction

This paper consists of two main sections. The first sections will focus on using Randomized Hill Climbing (RHC), Simulated Annealing (SA), and Genetic Algorithm (GA) as an alternative to Gradient Descent in optimizing the weights of the Neural Network trained on Heart Disease Dataset that was explored in assignment 1. In the second section Continuous Peak, Knapsack, and Flip Flop optimization problems are used to highlight the strength and weaknesses of RHC, SA, GA, and MIMIC.

II. Neural Network Weight Optimization

In this section, common optimization algorithms will be used to optimize the weights in a Neural network trained on the Heart Disease Dataset used in assignment 1. In particular, the algorithms that will be used in this section are Random Hill Climbing (RHC), Simulated Annealing (SA), and Genetic Algorithm (GA). The performance of the above-mentioned algorithms will be compared with Gradient Decent approach which is the most popular algorithm for weights optimization in Neural Networks.

A. Heart Disease Dataset (Recapitulation)

The Heart Disease dataset [1] used in assignment 1 will be utilized in this section. This dataset is provided by UCI machine Learning Repository [1]. It contains 14 features that are used to predict the possibility that a patient has a Heart Disease. Some common preprocessing techniques was used such as one-hot encoding for categorical value, and scaling. This is necessary since otherwise the neural network will be more sensitive to features with higher absolute values. For more details about the dataset, please refer to [1].

B. Methodology

The analysis in this report has been developed using mlrose package. The mlrose is a python package that contains common randomized optimization algorithms implementation. 75% of the Heart Disease dataset was used for training the Neural network while 25% was kept for testing. Cross-validation would be a better evaluation matrix than a single split. However, due to the limited time of the assignment, a single split evaluation was used. A neural network with one layer and 10 neurons was used in this section. This configuration was chosen based on the best-performing combination (number of layers and neurons) found in assignment 1. In addition, the best models of each randomized optimization algorithm were selected based on a grid search based on the testing accuracy.

Furthermore, since the Gradient Descent is the most common optimization technique used for neural networks, it will be used as the main benchmark for the other above-mentioned randomized optimization algorithm.

C. Benchmark - Gradient Descent

Gradient Descent is the most popular method for weight optimization of neural networks. It uses back-propagation to update the error based on the gradient of the error (difference in predicted and actual value). Since the hyper-parameters were already found using grid search in the previous assignment, the same parameters were used for this section. In particular, the learning rate is 0.01 with one layer and 10 neurons.

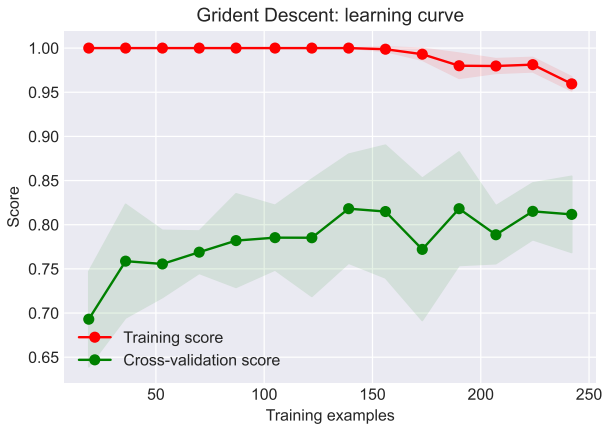

Fig. 1. GD learning curve

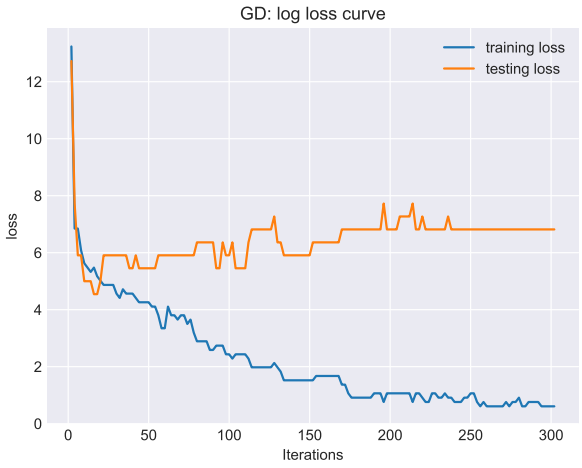

Fig. 2. GD Loss curve

Fig. 3. GD Train time per iterations

Figure 1, 2, and 3 shows the learning curve, loss curve, and training time per iterations for GD algorithm. These curves will be used to compare the performance of other randomized optimization algorithms (RHC, SA, and GA). The learning curve in figure 1 shows that the performance of the neural network shows some improvement as the number of samples increases. This means that Gradient Descent is able to improve the weight optimization with more samples. Similarly, the loss curve in figure 2 illustrates the speed at which GD minimizes the error reaching an error of about 4 in less than 20 iterations. This is expected since the GD algorithm updates the weights in the steepest direction which minimizes the error. However, it is also clear that after the 20 iterations, the model exhibits overfitting to the training data as only the training loss decreases while the testing loss started to increase. Furthermore, the training time per iteration is shown in figure 3. The figure shows that training time increases linearly with the number of iterations (with some variance). In addition, GD was able to perform 300 iterations in less than one second which is relatively fast compared to the other algorithms (RHC, SA, GA).

D. Random Hill Climbing (RHC)

Random Hill Climbing is an optimization algorithm that aims to find the optimum function (in this case, the weight of the neural network). It starts from a random point and iteratively climbs up the hill to a state with a higher value (in the case of maximizing). A common weakness of RHC is that it gets stuck in local maximum/minimum. To compensate for weakness, the algorithm has a restart parameter that allows RHC to start over several times to avoid getting stuck in a local optimum. Having said that, the restart parameter cannot guarantee that RHC will find the global optima, but it increases the probability of finding a better state. It should be noted that 5000 maximum iterations and 10 random restarts were used to produce the experiment for RHC (unless otherwise specified).

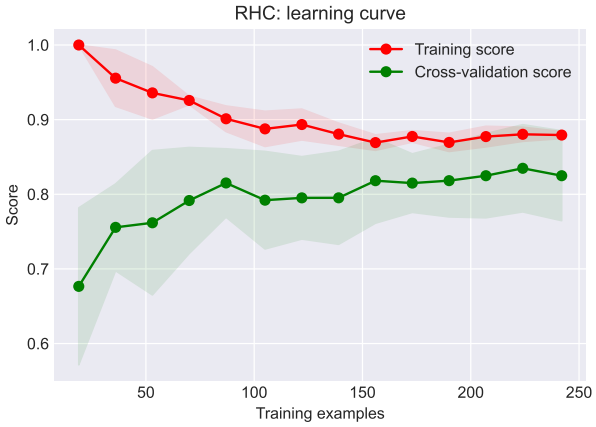

Fig. 4. RHC learning curve

The learning curve of RHC is illustrated in figure 4. The average cross-validation accuracy is similar to GD in figure 1 but with higher variance. This is expected due to the randomness of RHC. In addition, both GD and RHC show the utilization of more samples to improve the accuracy of the Neural Network. It is also interesting to observe in the case of RHC, the training accuracy decreased as the number of samples increased which means that the neural network with RHC was not able to fit the training data (underfit). This might be because of the stop criterion. In this case, the number of iterations is chosen based on the lowest loss of the testing split. However, the loss in the training set might still have room to reduce further, hence the training accuracy not always be very high.

Moreover, There are two important parameters in RHC: step size and restarts. Figure 5 shows the effect of both parameters on the accuracy. It is clear that the performance of RHC is greatly affected by the step size. The step can greatly affect the stability of the learning. Thus, it is important to tune the

step size. In addition, figure 5 shows that the performance of the NN was not greatly affected by the number of restarts. This may suggest the hypothesis that is learned by the Neural Network does not have many local optima.

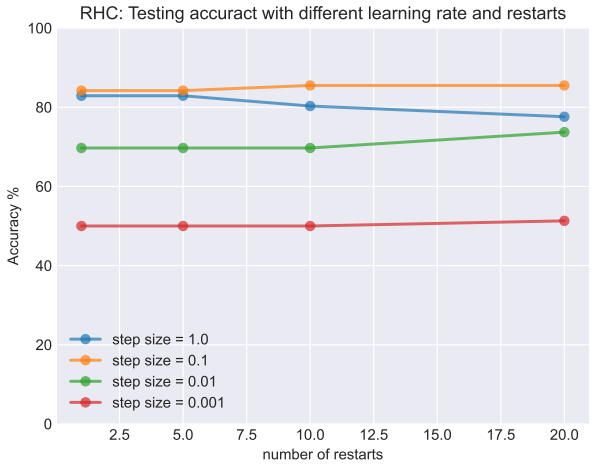

Fig. 5. RHC: Accuracy vs restarts

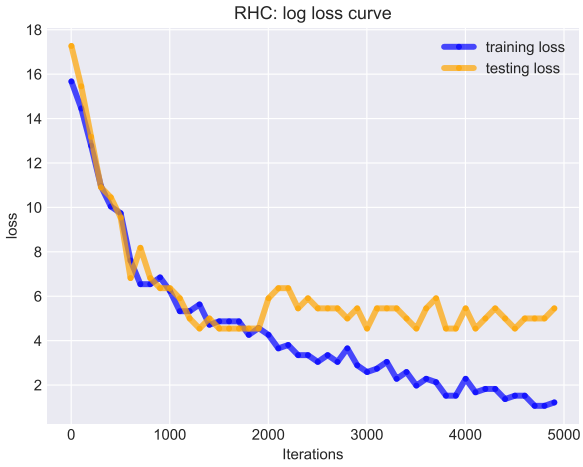

Fig. 6. RHC: Loss Curve

As stated previously, a drawback of RHC is that it is expected that it gets stuck in local optima. Thus, it requires multiple restarts which significantly increase the number of iterations compared to GD. In addition, RHC chooses to move in any random direction that improves the fitness function, unlike GD which updates based on the steepest direction that minimizes the error (through backpropagation). This also contributes greatly to making GD much more efficient than RHC. This can be seen in figure 6 which illustrates that RHC needs almost 50 times more iterations to achieve a similar loss as the GD. Thus, it is not surprising that RHC usually takes a much longer time to train since the number of iterations needed is significantly more than GD. This is also illustrated in figure 7. The training time also seems to be linear with the number of iterations plus some variance which is expected due to the inherent randomness of the algorithm.

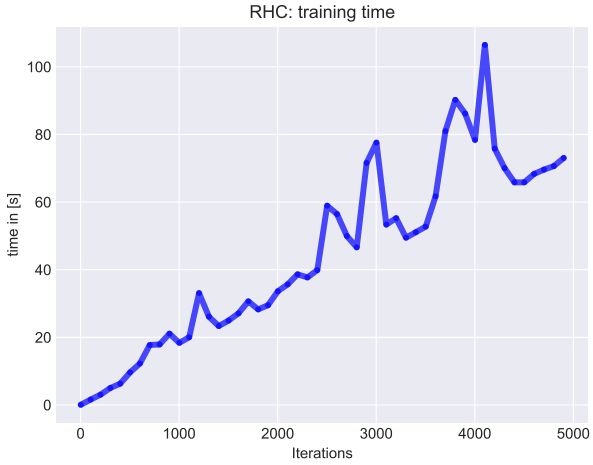

Fig. 7. RHC: Training time

E. Simulated Annealing (SA)

Simulated Annealing is an algorithm that was inspired by physical phenomena that involve both heating and cooling metals to change their propriety. Simulated Annealing is similar to RHC in the sense that it starts in a random place but it only moves to a better state with a certain probability. This probability is controlled by two parameters: temperature and decay rate. Typically, SA starts with high exploration (high temperature), then leans more towards exploitation (cooldown). This helps SA gets out of local optima that RHC can get stuck in. Once the temperature is very low, SA behaves like RHC. It must be noted that the experiment in this section was trained using 6000 iterations (unless otherwise specified)

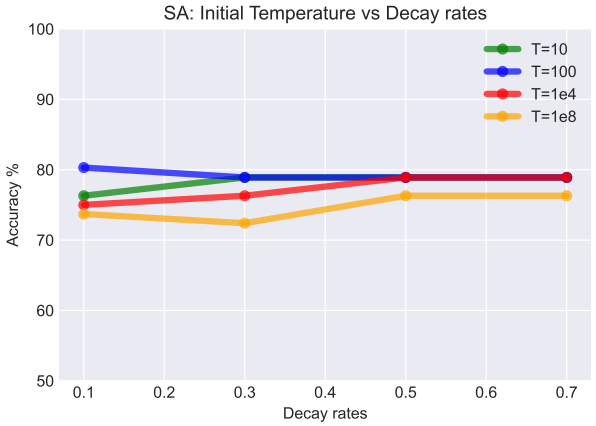

Fig. 8. SA: Accuracy Vs Decay rate

The temperature and decay rate parameters determine the exploration and exploitation trade-off. Figure 8 illustrate the performance of the different combination of starting temperature and decay rates. The figures show that the best combination of temperature and decay was 100 and 0.1 respectively. In addition, a similar result was achieved with higher temperature but also a higher decay rate. Since the randomness increases with high temperature, SA needs either more iterations or a high decay rate to have enough time to both explore and exploit the space.

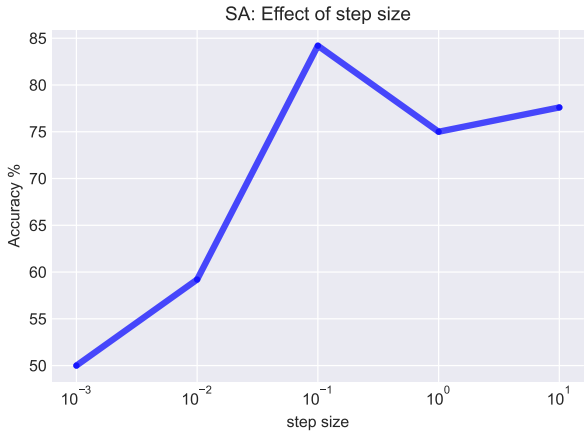

Fig. 9. SA: Accuracy Vs Step Size

In addition, similar to RHC, choosing an appropriate step size is very important in order to get good performance in any randomization algorithm. This is because a low step size usually leads to very slow learning, while a high step size leads to instability in training. This is illustrated in figure **??**. It can be seen from the figure the best step size for SA was 0.1 while much lower or higher values performed badly.

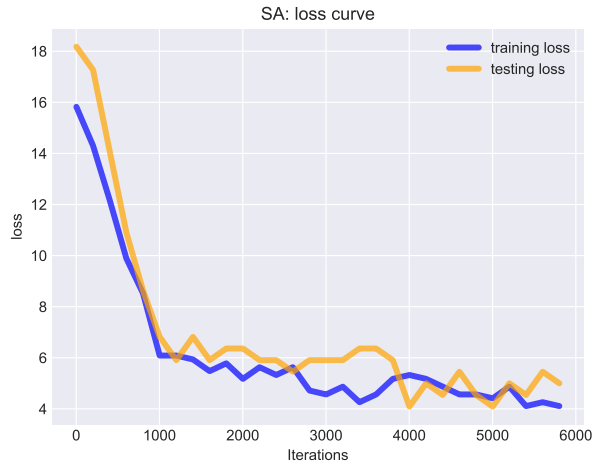

Fig. 10. SA: Loss curve

Fig. 11. SA: Training Time

The loss curve of Simulated Annealing is shown in figure 10. It can be seen that SA took almost 2-3 times more iterations than RHC to reach its lowest point. This is expected since SA spent a significant number of iterations exploring rather than exploiting. Another reason is that RHC has the advantage of restarting when it is stuck (10 restarts) which has the same objective as the temperature in SA but restarting is less efficient. In addition, although SA and RHC have a similar lowest loss, SA still achieved slightly a lower loss than RHC (around 4000 iterations). In addition, SA has slightly higher time complexity than RHC (per one restart) since there is an additional calculation of the temperature. This can be concluded by comparing figure 11 and 7. It must be mentioned that to compare the training time between SA and RHC, a division of 10 is needed in RHC since it is configured with 10 random restarts.

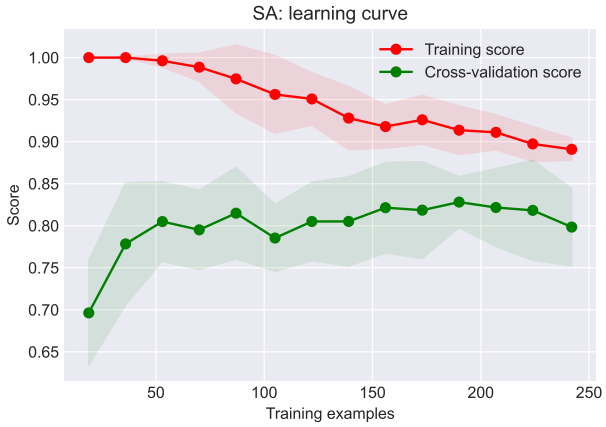

Fig. 12. SA: Loss curve

Figure 12 shows the learning curve of simulated annealing (SA) with a temperature of 100 and a decay rate of 0.1 that was trained with 4000 iterations. The 4000 iterations were

chosen based on the minimum loss value found in figure 10. It could be seen from figure 12 that the SA tends to produce a neural network that underfits with more training samples, unlike Gradient Descent (GD). However, it can still generate well to testing data with similar accuracy as the GD. This is expected since the best model is biased toward choosing the model with the lowest testing losses regardless if the training loss was at its minimum or not.

F. Genetic Algorithm (GA)

Genetic Algorithm (GA) was inspired by biological evolution where genes mutate to adapt to new environments. GA starts with a certain number of populations and iteratively produces a new population with mutated states then only keeps the highest performing states in the populations.

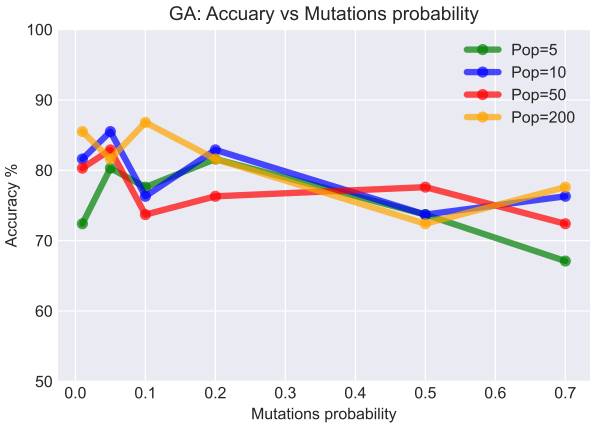

Fig. 13. GA: Accuracy vs Mutation Probability

Genetic Algorithm (GA) has two important parameters that should be carefully tuned: population size and mutation probability. GA starts a fixed **population**. Each state in the population consists of elements (Chromosomes). In each generation (iteration), these Chromosomes have a certain probability to mutate to produce new states in the population. Figure 13 illustrates the effect of varying the population size and mutation probability on the accuracy of the neural network. It is shown that the highest population (200) and a mutation of 0.1 produced the best result. This is intuitive since a higher population has a higher chance of reaching the optimum state. In addition, a mutation of 0.1 seems reasonable since a higher mutation rate could produce many changes at once which increases the randomness of the search. This could be clearly observed from figure 13. At a higher mutation rate, most population sizes tend to perform worse than when the mutation rate is about 0.1-0.3. It is also interesting to see that a higher population will not always produce better results because of the inherent random of GA (mutation rate).

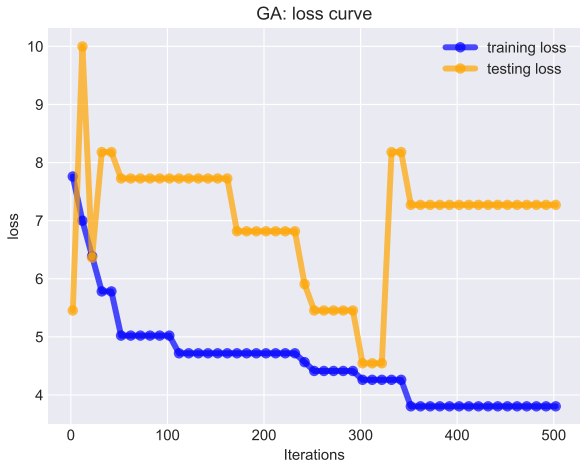

Fig. 14. GA: Loss Curve

The loss curve of GA is shown in figure 14. It can be seen that the training loss decreases as the number of iterations increases. This is expected since the GA takes the fittest states in each generation. It is also interesting to see that training loss was constant for multiple intervals meaning that GA was not able to find better states for several generations. Moreover, GA exhibited overfitting after around 300 iterations as the testing loss started to increase while the training loss decreased.

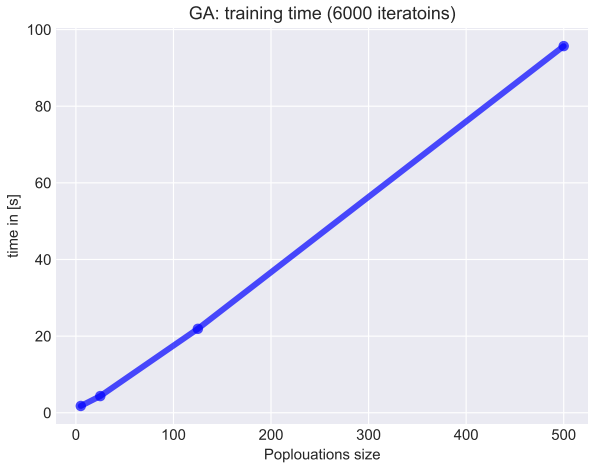

Fig. 15. GA: Training Time

Although GA has similar accuracy to GD, it takes significantly more time. Figure 15 shows the time complexity of GA. It can be observed that the time complexity is linearly proportional to the population.

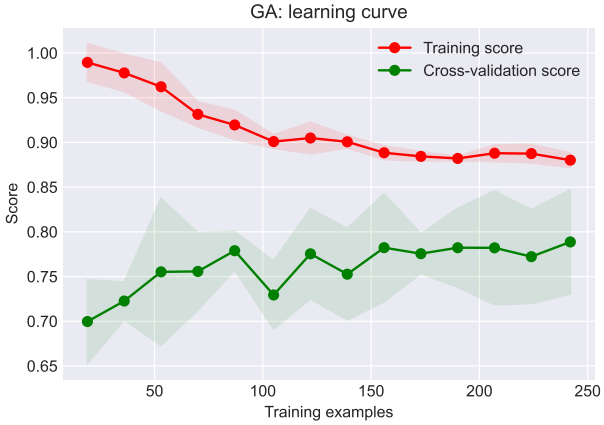

Fig. 16. GA: Learning curve

In addition, the learning curve for GA is shown in figure 16. GA exhibited a similar pattern for both the training and cross-validation curves compared to the other randomized optimization algorithms. The figures show that the more that the harder for GA to find optimal weights for NN based on the training data. Despite this, GA produced similar average cross-validation accuracy compared to the other randomized optimization algorithms.

G. Conclusion

TABLE I
NEURAL NET WEIGHT OPTIMIZATION SUMMARY

Algorithm	Accuracy %	Num Iterations	Training Time [s]
GD	87	18	0.08
RHC	87	1500	114
SA	88	4000	13
GA	87	300	64

Table I shows the summary of this section. The result shows that all examined randomized optimization algorithms produced very similar overall accuracy. However, the time complexity of the algorithms is very different. The Gradient Descent (GD) is by far the faster algorithm, and it also takes significantly less number of iterations to produce good accuracy. This is expected since the GD always takes a step toward the steepest state that minimizes the error. It can be also noticed that SA was the second fast algorithm but has the highest number of iterations. It should be mentioned that the RHC is configured with 10 restarts, and hence took much more time than SA.

III. Optimization Problems

This section is focused on implementing three optimization problems that highlight the strength and weaknesses of four randomized optimization algorithms: RHC, SA,GA, and MIMIC. The selected optimization problems are Continuous Peaks, Knapsacks, and Flip Flop. In addition, the algorithms are considered converged if the algorithm exhausted five failed attempts. This means that in each iteration, the algorithm has a maximum of five attempts to find a better state than the current best state. The reason only five attempts are allowed is to test which algorithm gets stuck faster than the others which can highlight the strength and weaknesses of such algorithms.

Regarding the parameters of each algorithm, RHC is configured with 20 restarts, SA with a temperature of 100, and a decay rate of 0.1. In addition, the higher the population for GA and MIMIC, the higher chance that the algorithm produces better fitness values. However, in this paper, the population is set to be 500. In addition, the keep percentage for MIMIC was set to be 50%. These parameters are found using the grid Search approach.

A. Continuous Peaks

This optimization problem is an expansion to the four-peaks and six-peak problems. The Continuous-Peaks problem consists of many local maxima. This will be particularly challenging RHC as there is a high chance that RHC will get stuck in one of the local maxima. This optimization problem will highlight the advantage of using Simulated Annealing (SA) which is capable of escape of these local maxima and eventually reaching the global optima. It will also expose the weakness of RHC since it is likely to get stuck in local maxima. MIMIC and GA will likely perform well for small states size but it also likely to perform badly as the sizes of the problem increase.

The Continuous Peak is configured to accept sequences of 0s and 1s. The fitness value is given based on the maximum number of consecutive 0s or 1s. There is also a bonus to the fitness value if both continuous sequences of 0s and 1s are more than a certain percentage of the state's size (T). In this problem, the threshold T is set to be 20% of the number of bits. For example, if the state is given by [0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1], and T is 15%, then the fitness value will be 17. Moreover, the optimum value of the continuous peaks value is 2N-T-1, where N is the number of bits, and T is the threshold.

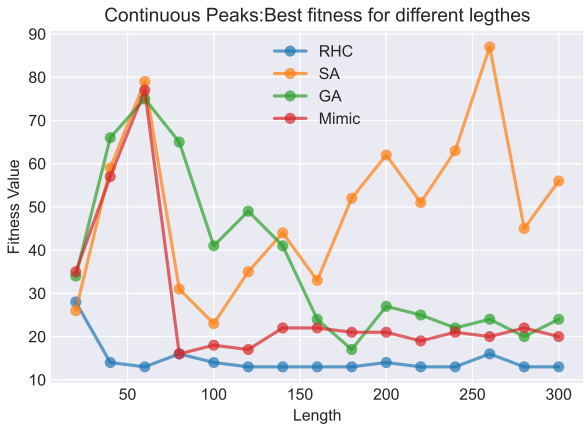

Fig. 17. Continuous Peaks: Fitness vs Length [Max attempts = 5]

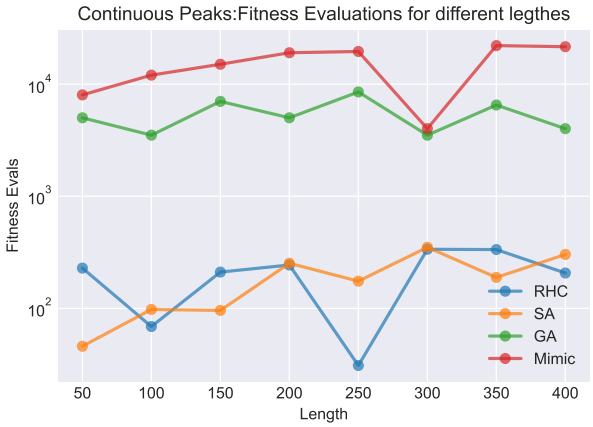

Fig. 18. Continuous Peaks: Fitness Eval vs Length

Figure 17 shows the best fitness value obtained from each algorithm at different state lengths. It is clear that SA showed superior performance compared to the other algorithms. This is expected since SA does not get stuck in local maxima but rather it keeps exploring until its temperature cools down. It is also interesting to see that SA was not among the highest in the number of fitness function evaluations as it is seen in figure 18. Both GA and MIMIC are configured with a population of 500 which means that in each iteration they need to evaluate the fitness function 500 times which is the reason why they have a high number of function evaluations compared to the other algorithms.

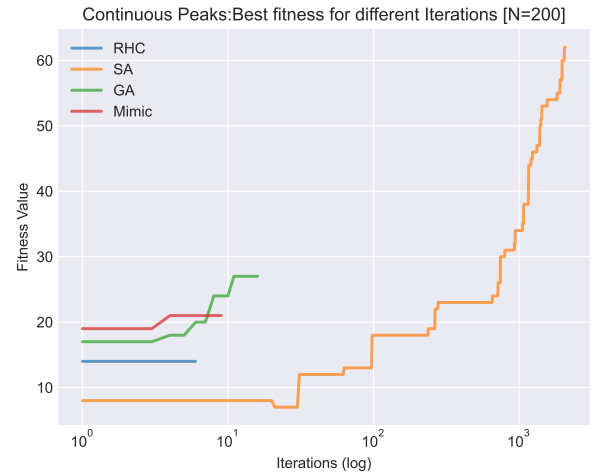

Fig. 19. Continuous Peaks: Fitness Eval vs Length

To illustrate the progress curve of the algorithms, I have implemented an experiment on the Continuous Peaks problem with a length of 200. Figure 19 illustrates the fitness value per iterations. The graph clearly shows that all algorithms get stuck and converge early except SA which continues exploring for higher maxima. It can be seen that even with 20 restarts, RHC still get stuck in local maxima, whereas the temperature concept in SA helped to avoid and find higher fitness value than all other randomized optimization algorithms. GA algorithm performed poorly for higher state sizes. Given that 5 maximum attempts per iteration are given for each algorithm to find a better fitness value, GA is expected to struggle in finding absolute maxima and is likely to get stuck early. In addition, MIMIC has a similar performance to GA. Since this optimization problem has many local maxima, the MIMIC is likely to create a uniform distribution, and therefore get stuck and converge early as well.

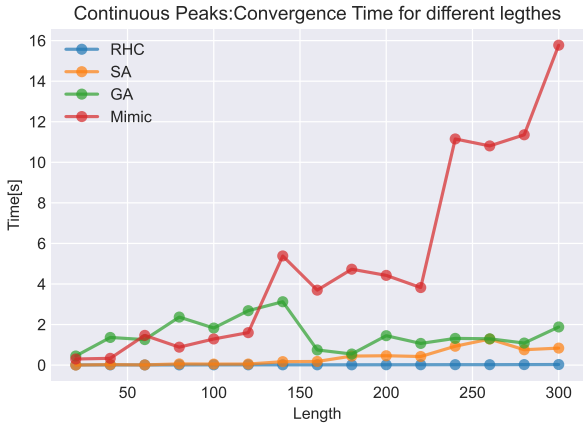

Fig. 20. Continuous Peaks: Time complexity

In addition, although SA takes significantly more iterations to converge, its time complexity and number of fitness evaluation is relatively low compared to GA and MIMIC as it is shown in figure 20 and 18. This is expected since SA and RHC are simple algorithms compared to GA and MIMIC.

B. Knapsack

The knapsack is another common optimization problem. The Knapsack problem consists of items that have both values (v) and weights (w). Knapsack aims to maximize the value of the items given a weight constraint. This problem will highlight the advantage of using GA as well as exposes the weaknesses of RHC and SA. One reason why RHC and SA do not perform compared GA and MIMIC is that the fitness function in this problem returns 0 once the allowed weights are exceeded. This can easily confuse both RHC and SA since their search is in a random direction. The randomness of RHC and SA makes it likely to get fitness values of 0s and therefore terminates early.

On the other hand, once GA finds a generation with reasonable fitness, it continues building on this generation to produce fitter states (generations). This can be observed from 21 which illustrates the superiority of GA compared to the other algorithms. It must be noted that MIMIC also has reasonable performance since it is similar to GA algorithm. However, MIMIC usually are good in problems with a well-defined structure which could be the reason why GA performs slightly better in this problem.

Fig. 21. Knapsack: Fitness value for different number of items

The higher performance of GA is at the cost of higher number of fitness evaluation it is shown in figure 22. However, the difference between the number of fitness evaluation between GA, MIMIC and SA is relatively low. If the fitness evaluation is expensive, then MIMIC will be slightly preferred in this case.

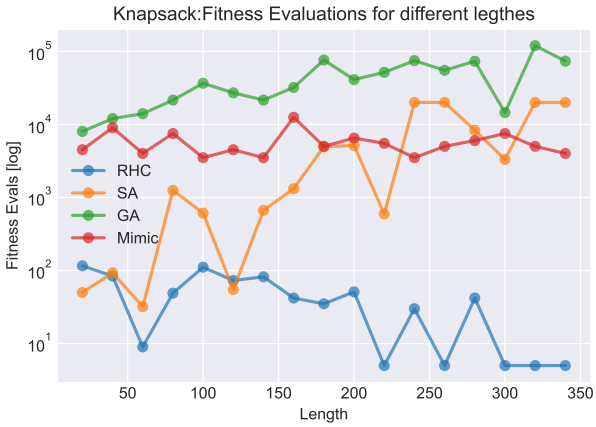

Fig. 22. Knapsack: Fitness Eval for different number of items

An example of the progress of the fitness value per iteration is shown in figure 23. The graph looks a little odd since SA spent many iterations exploring the space before it converge compared to the other algorithms. In addition, it can be seen that MIMIC and GA needed significantly less iterations to provide a reasonable fitness value.

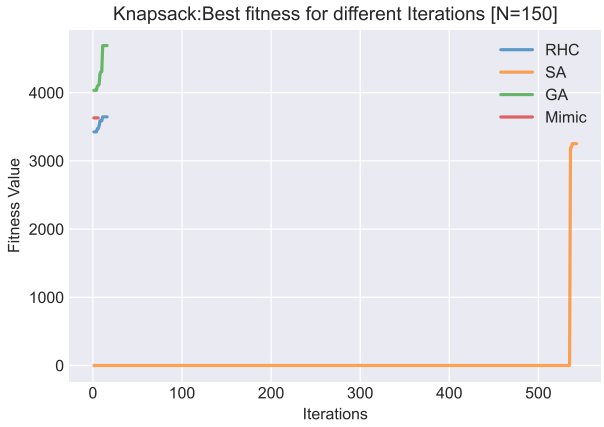

Fig. 23. Knapsack: Fitness per iterations [N=150]

Although GA and MIMIC have superior performance in the Knapsack problem, they have higher time complexity than SA and RHC. This complexity difference becomes more significant as the population size increases. This can be seen from figure 24 which shows that MIMIC has the longest time to converge followed by GA.

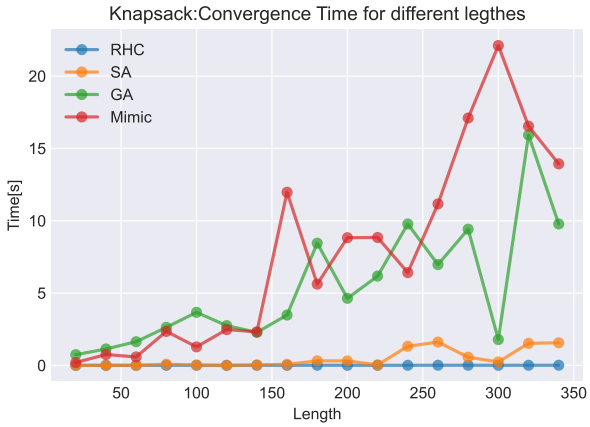

Fig. 24. Knapsack: Convergence Time vs Number of Items

C. Flip Flop

Flip Flop is an optimization problem that aims to maximize the alternation between the bits in the state. It accepts a discrete state of 0s and 1s. Its fitness value is calculated based on the number of different consecutive pairs (alternations). Although the strength of MIMIC is partially shown in the Knapsack problem, this problem will also highlight the advantage of using MIMIC when the fitness function has a specific structure (alternation or chain in this case). Since this is a relatively simple problem, all the algorithms are expected to perform well if they have a high number of attempts per iteration. However, to demonstrate the superiority of MIMIC in this example, the maximum attempts to find a better fitness value were limited to only five attempts.

Since MIMIC is the only algorithm that was not explored in the weight optimization section, its hyper-parameters will be briefly explored.

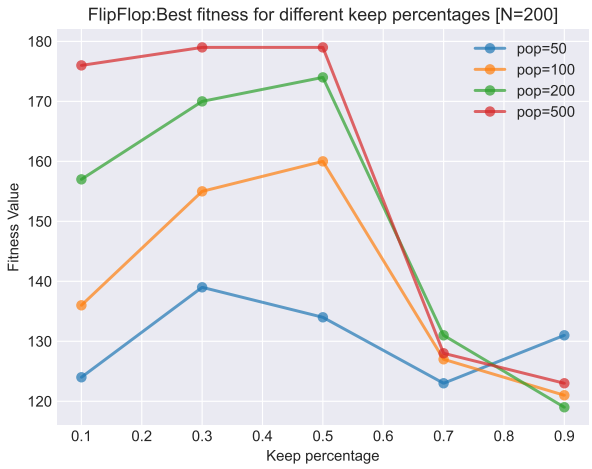

Fig. 25. Flip Flop: Effect of population size and keep percentage on fitness value

MIMIC has two important parameters: population size and keep percentage. Figure 25 shows the effect of varying the population size and the keep percentage. It is clear from the figure that a higher population increases the performance of MIMIC since a higher population increases the probability of reaching the global optimum (more search). However, what is interesting is a keep percentage higher than 50% seems to significantly deteriorate the performance of MIMIC. This could be because higher keep percentages hold up many states with low fitness values and hence slow down the optimization. In this section, the MIMIC algorithm was configured with a 500 population size and 50% keep percentage.

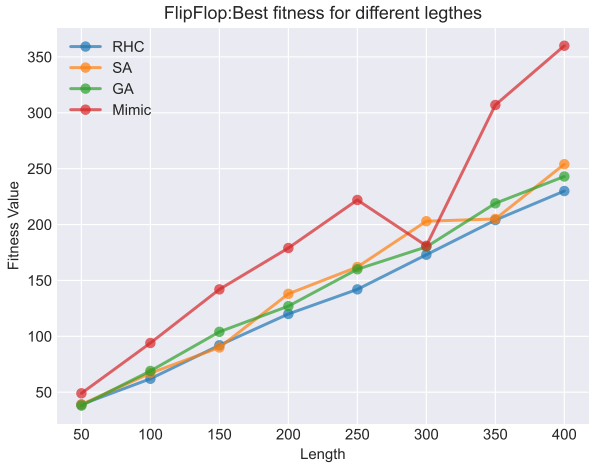

Fig. 26. Flip Flop: Best fitness vs length

An indicative experiment to evaluate the performance of the randomized optimization algorithms on the Flip Flop problem can be shown in figure 26. The figure illustrates the best fitness value for different Flop Flip problem sizes. It can be observed that MIMIC has outperformed all the other algorithms at almost all the sizes illustrated in the figure. This is not surprising since MIMIC typically works well with fitness functions with fixed structures (chain of alternation 0s and 1s). Other algorithms are likely to get stuck early and terminate since they randomly search the states instead of looking for a specific structure, unlike MIMIC.

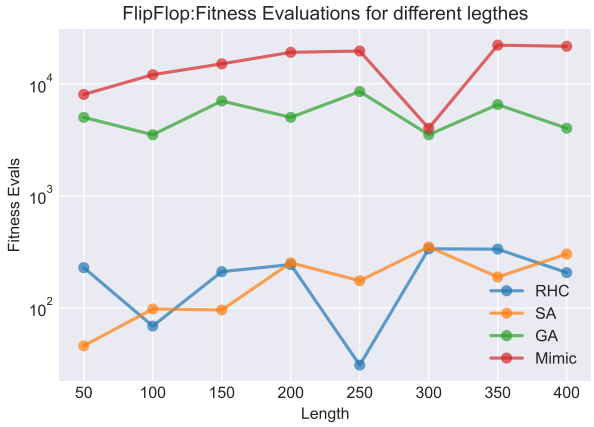

Fig. 27. Flip Flop: Fitness Eval for Different Problem Lengths

Furthermore, figure 27 shows that MIMIC has the highest number of fitness evaluations. This is because MIMIC evaluates the fitness function 500 times (population size) per iteration. Thus, it seems the cost for higher performance of MIMIC is having higher fitness evaluations. This might be an issue if the evaluating fitness function is expensive. In this case, SA or RHC could be used with higher maximum attempts since they have much lower fitness evaluations than GA and MIMIC.

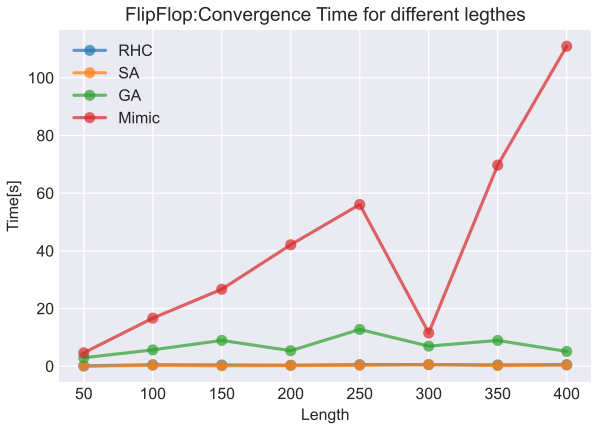

Fig. 28. Flip Flop: Best fitness vs length

It is also not surprising that MIMIC has a higher time

complexity than other algorithms. This was shown in previous sections and can be also seen in figure 28. If the problem is used in a real-time application, MIMIC might not be the best option. Faster algorithms such as SA could be used although it will be a trade-off between time and performance.

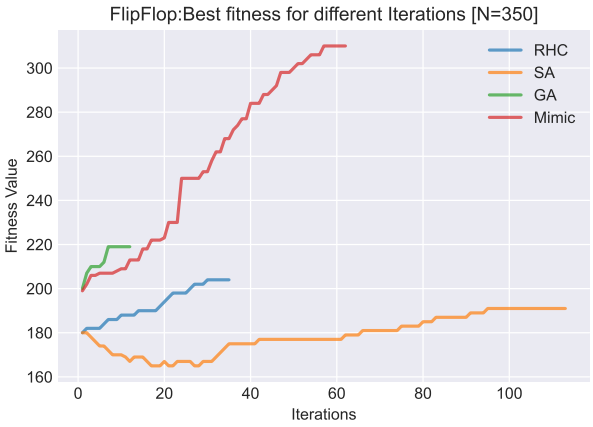

Fig. 29. Flip Flop: fitness per iteration

The optimization curve for length 340 is shown figure 29. The figure shows that both GA and RHC got stuck quickly in local maxima, while MIMIC was able to continue improving the fitness value and convergence after around 60 iterations since the fitness function contains some structure that helps MIMIC find better fitness values.

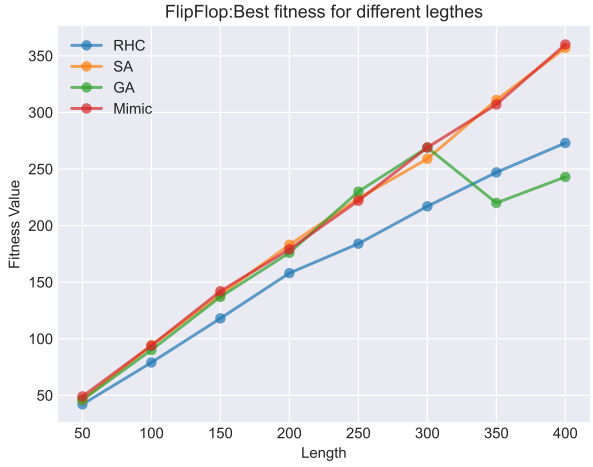

Fig. 30. Flip Flop: fitness per iteration

It must be mentioned since the Flip Flop is a relatively easy problem, all the algorithms can perform pretty well if given high maximum attempts per iteration. For example, figure 30 shows the performance of the algorithms with maximum attempts to find better fitness per step was limited to 20 instead of 5. Figure 30 that all algorithms have a similar performance which eliminates the superiority of MIMIC given a high number of attempts is allowed.

IV. Conclusion

The first section of this paper focused on comparing RHC, SA, and GA to Gradient Descent (GD) in optimizing Neural Networks weights. It was concluded that given the randomized algorithms (SA, RHC, GA) are given enough attempts, they can produce an NN model with similar accuracy as the GD. However, GD takes significantly less time to produce good results than the other randomized optimization algorithms. Another observation that is worth mentioning is that RHC, SA, and GA seemed less prone to overfitting may be due to their stochasticity nature.

In addition, the second section introduced three optimization problems that aim to highlight the strength and weaknesses of SA, GA, and MIMIC. The continuous peaks problem shows the advantage of using SA compared to the other algorithms. Since the continuous peak problem has many local maxima, SA has the advantage of avoiding getting stuck in local maxima, unlike the other algorithms. In the Knapsacks problem, It was shown that GA outperforms the other algorithms. The reason that the other algorithms perform poorly in knapsack is that the fitness value can suddenly drop to 0 once the allowed weight is exceeded. However, since GA keeps the fittest population, mutations of a new generation can only produce states that are higher or the same (in the worse case) but never worse. Finally, MIMIC was shown to perform the best in the Flip Flop problem. This is due to the fact that MIMIC looks for structure in the fitness function, which is presented in Flip Flop (alternations).

References

[1] Andras Janosi et al. *Heart Disease Data Set*. 1988. URL: https://archive.ics.uci.edu/ml/datasets/Heart+Disease.