Randomized Optimization

CS7641: Machine Learning

Kim, Irene
skim3364@gatech.edu

I. Objective

The objective of this report is to analyze different search techniques in-depth. The first part of the report will compare the behaviour of different randomized optimization algorithms on three discrete optimization problems. The second part of the report will use three algorithms in neural network and compare the result with the backpropagation used in the previous assignment[1]. The remainder of the paper is structured as follows: Section 2 provides a theoretical framework for the four different optimization algorithms. Section 3 briefly describe three discrete optimization problem selected for this report. This section also compares and analyzes the result of using different search algorithms on the chosen problem. Section 4 compares the neural network with three optimization algorithms and the neural network with backpropagation. The same dataset from assignment is used in the evaluate the result. Finally, the work is summarized in section 5.

II. Optimization Algorithms

A. Random Hill Climb Search

Random hill climb search(RHC) is a variant of hill climb search algorithm. Hill climb search keeps track of only the current state and moves to the neighboring state with highest value one iteration at time. RHC, instead, conducts a series of hill climb searches. RHC search begins with randomly generated initial state and terminates when goal is found. Thus, it speeds up the search process. It also has less chance to get stuck in local maxima compared to the standard hill climb search. The RHC can quickly find a good solution when the problem has only one or few local maxima and plateaus[1].

B. Simulated Annealing

The problem with the hill climbing algorithm is that it never backtracks. To be specific, the hill climbing algorithm is prone to get stuck in a local maximum as it never makes downhill movements. Simulated Annealing(SA) overcomes the drawback of hill climbing methods by using a temperature term. The SA starts with a high temperature and then slowly cools down the temperature using the following property of Boltzmann distribution:

$$e^{\frac{\Delta E}{T}} \tag{1}$$

The SA picks a random move based on temperature property. When the temperature is high, it allows the algorithm to accept even bad movements. When the temperature is low, the e becomes a very small number that there is no chance of taking a new suggested position. Thus, when the temperature is very small, it becomes just a normal hill climbing. The intriguing trait is that it will still take the move that is worse than the current point with probability $e^{\frac{\Delta E}{T}}$. The SA guarantees to converge to the global maximum as long as it starts with a high enough temperature and decreases the temperature slow enough.

C. Genetic Algorithm

The genetic algorithm (GA), also known as the evolutionary algorithm, imitates a human's biological evolution procedure. GA goes through a phase of selection, crossover, and mutation to generate population. First, the initial population is evaluated on a predefined fitness function. The best hypothesis is the one that optimizes the hypothesis's fitness. Based on the fitness score, each hypothesis receives a probability that the current hypothesis will be selected. The crossover phase is the recombination procedure that randomly selects a crossover point to split parent stings. This permutation process can help produce a meaningful combination of strings as long as the sub-strings contain meaningful components. Finally, the mutation process flips every bit in its composition, introducing some randomness. The whole process is repeated until convergence. The GA replaces a parent hypothesis with an offspring that may be radically different from their parents. The choice of the fitness function has a significant impact on overall performance.

D. MIMIC

The MIMIC algorithm is a search technique that directly models probability distribution. It successively refines the model over time that will, eventually, convey structure and parts of space that are more optimal than other points. The MIMIC estimates the distribution by finding the dependency tree. Assuming that the best tree we can find is the one that maximizes the information gain, the MIMIC utilizes the

[1]The code is available at GitHub: https://github.com/skirenekim/CS7641-Machine-Learning/tree/main/HW2

maximum spanning tree as the dependency tree. Because the MIMIC captures both the history of its search and probability distribution, it works well with problems that have structures or relationships. However, the MIMIC is very expensive to compute each iteration as you get more information at the cost of estimating probability distribution such as maximum spanning tree.

III. Three Problems

The three chosen problems for this report are the flip flop problem, the four peaks problem, and the Knapsack problem.

A. Flip Flop

Given a bit string, the flip flop problem is finding the total number of consecutive bit alternations. The optimal configuration of bits string in the flip flop problem would be a big string consisting of continuously alternating bits. The global optima, in this case, would be N-1. Although simple, the flip flop problem is an interesting optimization problem as it has a large search space to find a global optimum. Therefore, an algorithm that has less chance to get stuck in local minima in the large search space may perform well in this problem. Figure 1 compares different optimization algorithms and their fitness score concerning the size of inputs or the problem size. It shows that SA performed the best with the flip flop optimization problem. Figure 2 suggests that SA also took significantly less time compared to GA and MIMIC. Further analysis and comparison of each algorithm are continued in the below subsections.

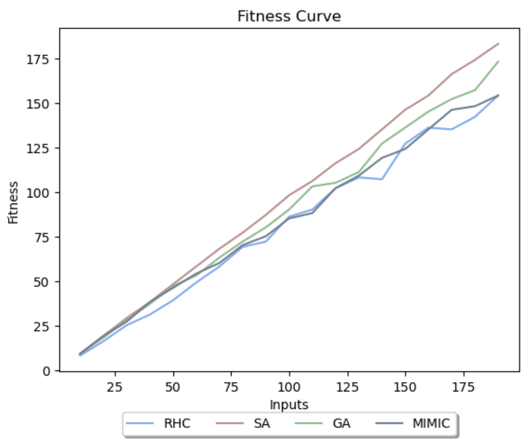

Fig. 1. Result Comparison

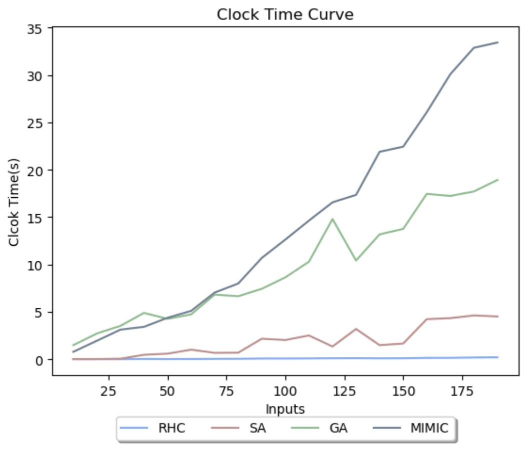

Fig. 2. Result Comparison

RHC

The RHC took the least time compared to the other algorithms. As it never backtracks and only moves upwards, the computation cost is significantly lower than GA and MIMIC. The RHC has a restart hyperparameter. This value controls the number of random restarts. The default value is 0. Figure 3 illustrates different fitness scores based on the number of restarts used in the RHC. The restart value of 10 converged fastest within 150 iterations, reaching slightly above 80 for the fitness score. Nevertheless, there were no noticeable improvements among different hyperparameter values. Therefore, we could assume that alternative hyperparameters also might not result in a huge performance boost for this optimization problem. One reason to explain the worst result is that RHC greedily grabs a better neighbour state. Although random restart alleviates this behaviour by exploring random spots, the problem we have contains many local maxima. Therefore, the chance of getting stuck at local maxima is greater than in other algorithms. To overcome this major weakness of RHC, we can use SA, which allows the algorithm to move more randomly.

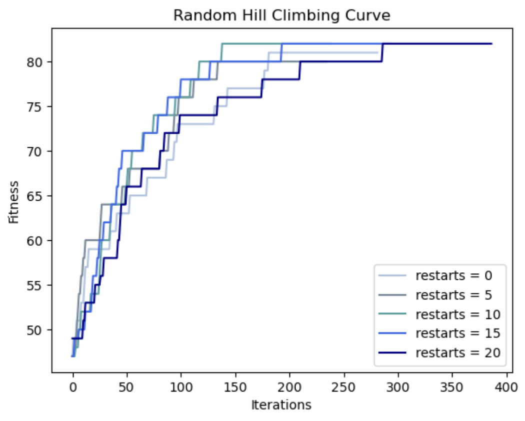

Fig. 3. RHC Hyperparameter Comparison

SA

As expected, the SA fitness score reached almost 100, which wins all the other three algorithms. SA use the temperature to allow more randomness in the early stage of training. The temperature is gradually reduced, which rejects randomness and takes hill-climbing movement until convergence. This mechanism can help SA find global maxima, especially problems with many local maxima like the flip flop problem. The decreasing temperature method is the hyperparameter for SA. Three different scheduling methods were compared to determine the value of the temperature parameter. Exponential converged quickest in this case with the problem size of 100. However, all three temperature-reducing methods showed only small differences, indicating that the SA performs well on this optimization problem less sensitive to the hyperparameter choices. The SA not only accomplished the best fitness score but also achieved the best result with a relatively lower computational cost. It still takes much less time than GA or MIMIC as does not store any previous state. Thus, the overall result conveys that SA is the best choice for optimizing the flip flop problem.

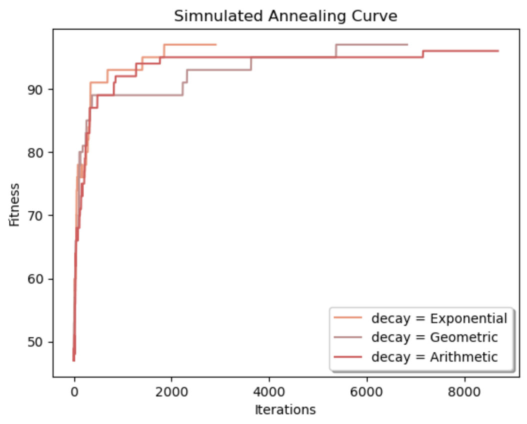

Fig. 4. SA Hyperparameter Comparison

GA

The GA performed best after the SA, reaching slightly above 95 on the fitness core. The GA's ability to search complex hypotheses spaces may have contributed to a good result. The mutation probability is the probability of a mutation at each element of the state during the mutation phase. The population size is the size of the population used in GA. Figure 5 presents that GA with a mutation probability of 0.3 and population size of 500 reached the highest fitness score. Generally higher mutation probability led to a better result. The mutation introduced randomness into the algorithm. The randomness allows the algorithm to explore the areas with higher fitness scores that are previously unexplored with the parent population.

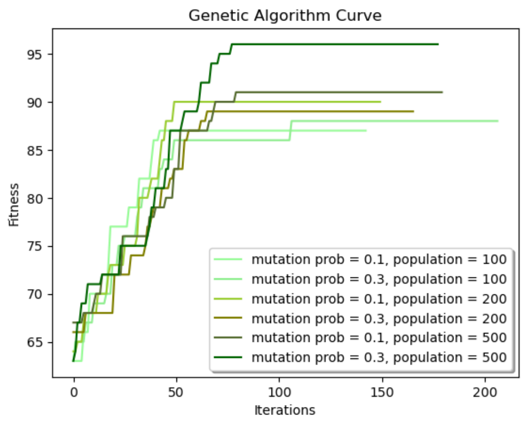

Fig. 5. GA Hyperparameter Comparison

MIMIC

The MIMIC reached slightly above 95 on the fitness score with its best hyperparameter choice. The MIMIC performed only slightly better than RHC but took a significantly longer time to compute. It also took the longest time to converge compared to other optimization algorithms. Therefore, MIMIC would not be a proper method to use when there is a time constraint and when the search space is very large. The hyperparameters explored are population size and proportion of samples to keep at each iteration of the MIMIC. The best values are 0.3 for the keeping samples proportion and 500 for the size of the population. Figure 6 suggests that a bigger population size tends to lead to a better result. This is because having a larger population size enables predicting a more concrete underlying probability distribution of the data. In contrast to the hyperparameter configuration with the lowest fitness score, the best hyperparameter configuration output approximately 15 percentage points higher fitness score. The algorithm, therefore, may have further room for improvement by having a large population size and a proper keeping percentage. However, the time it takes for each iteration is quite expensive. The number of iterations to reach the best fitness score was around 20 iterations. MIMIC usually takes fewer iterations to converge, but each iteration takes a long time. Overall, the result represents that MIMIC would not be an efficient algorithm for flip flop problems.

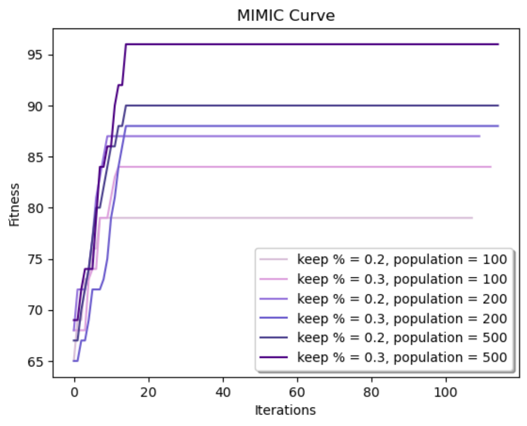

Fig. 6. MIMIC Hyperparameter Comparison

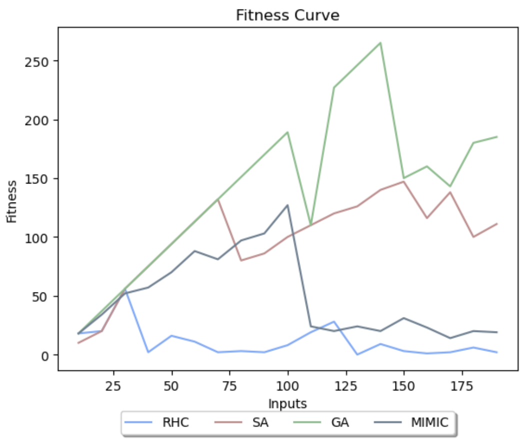

Fig. 8. Result Comparison

B. Four Peaks

The four peaks optimization problem contains two local optima within basins of attraction in a 1D space. The problem contains two small peaks that contain lots of 0s or 1s. Two larger peaks have larger bonus values. The optimal solution for this problem is to output a fitness score with maximum values. Specifically, an algorithm reaching the bonus points can produce the best solution possible for the four peaks problem. The four peaks problem is interesting as it resembles many real-world optimization problems. There are many good solutions, but we wish to find the optimal solution, which is the larger peaks with bonus points. The results can easily be between different algorithms. Comparing the result of the four optimization algorithms, the GA reached the best solution followed by SA. Figure 7 depicts the fitness score and figure 8 shows the computation time. Four peaks problem closely resembles a human chromosome configuration. Therefore, picking the best bit string from the parent and performing the crossover and mutation step might have resulted in GA producing the most elite off-springs to solve the problem.

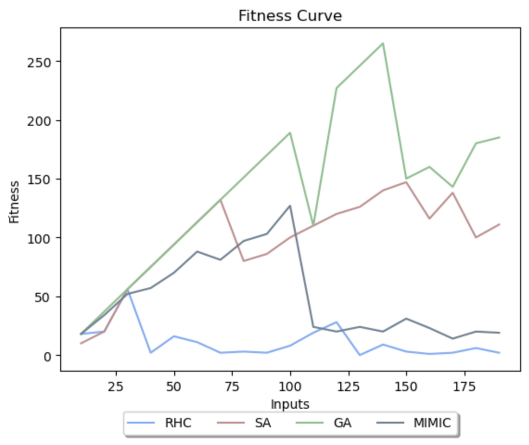

Fig. 7. Result Comparison

RHC

The RHC performed worst for the four peaks problem. Trying different numbers of restarts did not improve the performance. The downside of the RHC is that it greedily searches for the best value. In the worst case, if all the restarting points started at a similar point, all of them will not score well. Moreover, the problem of RHC is that in space with many local optima, although it is guaranteed to converge to the local optimum, converging to the global optimum is not guaranteed. Figure 9 portrays how the fitness score changes with a different number of restarts. One interesting result from the figure is that when the number of restarts was 5 it reached the best result. This suggests that having a large number of restarting points can have a greater chance of starting at a good search spot at the cost of more time, but sometimes less number of restarts can generate a better result. What this conveys is that due to the RHC's dependence on randomness, it can often lead to an arbitrary result. If we are hoping to use a more reliable and stable result-generating algorithm, RHC is not the optimal choice.

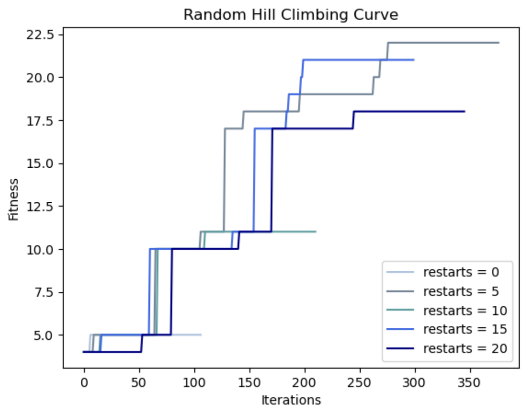

Fig. 9. RHC Hyperparameter Comparison

SA

The SA is the second-best optimization method with the four peaks problem. SA can jump off from the local maximum easily, which can explain its good performance. Three different weight-decreasing methods were experimented as can be seen in figure 10. The exponential decay reached the best fitness score the fastest followed by geometric decay. All three methods reached a similar result, but the arithmetic decay took the longest iterations to converge. Decreasing the temperature is important for SA because it controls how much time it will give the algorithm to explore the search space or how fast it should converge. Thus, we can assume that the arithmetic decay allowed the algorithm to wander around longer to search space before it converges.

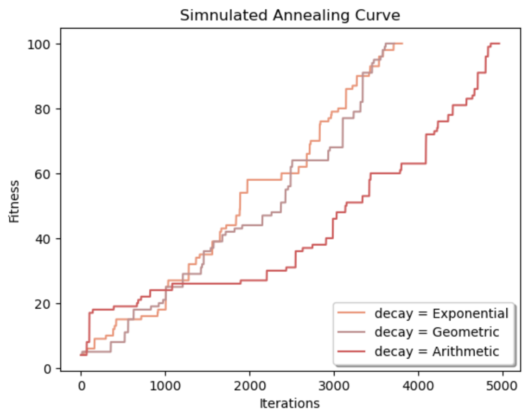

Fig. 10. SA Hyperparameter Comparison

GA

The GA is the best-performing algorithm for this problem. As briefly mentioned above, this optimization problem contains a bonus point. The optimization algorithm that reaches this bonus point will produce the best solution. GA with a mutation probability of 0.3 and population size of 500 performed well compared to other configurations tested. The mutation rate and population size have a big impact on the result as can be seen in figure 11. The larger population size generated a higher fitness score compared to the smaller-sized population in general. Having a larger pool of population encapsulates a more diverse population that can help produce more superior off-springs. In addition, a higher mutation rate produced better results than a lower mutation rate. This comparison illustrates that introducing randomness with large samples created elite and novel off-springs that can solve the optimization problem. Moreover, the large gap between different hyperparameter choices shows that with better hyperparameter options, we might be able to find a set of hyperparameters that produce even better results. If more time and resources were provided, it would be nice to see how the performance changes with respect to the mutation probability while fixing the population size as big values such as 500. This attempt may be able to reveal more about the general behaviour of GA on mutation probability.

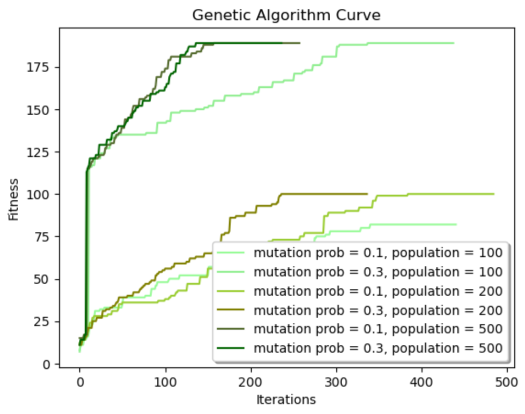

Fig. 11. GA Hyperparameter Comparison

MIMIC

Again, the MIMIC took significant computation time compared to other algorithms but still produced a good solution. Learning the four peaks problem's underlying probability distribution can enable MIMIC to perform well. Similar to the GA, MIMIC tend to output higher fitness score with a larger population size. The proportion of samples to keep at each iteration of the algorithm was better with 0.2 than with 0.3. The interesting part of the result in figure 10 is that there is a large discrepancy between different hyperparameter configurations. For example, compared to the MIMIC with a keeping percentage of 0.3 and population size of 100, the MIMIC with a keeping percentage of 0.2 and population size of 500 resulted in a fitness score more than 7 times higher. This reveals that choosing other values of hyperparameters carefully can result in further improvements in the performance than what we have now. Similar to the analysis of GA above, it would be nice to experiment with a grid search of keeping percentage while fixing the population size to see whether a big keeping percentage is more desirable or a smaller keeping percentage leads to a

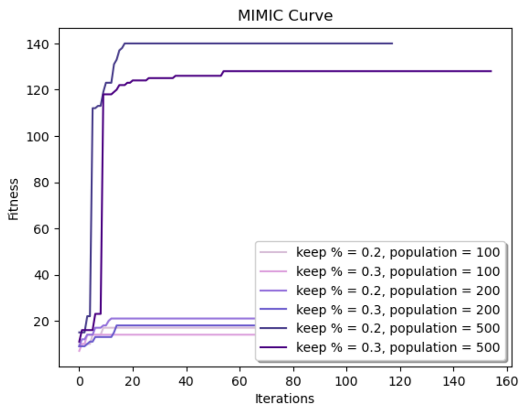

Fig. 12. MIMIC Hyperparameter Comparison

C. Knapsack

The Knapsack problem involves a set of constraints that given a number of values and the weights of each object, the algorithm needs to maximize the occupying volume in a knapsack without exceeding a specified threshold weight limit. In other words, the Knapsack optimization problem needs to determine the optimal configuration of the total weight and number of an object given the weight constraint. This optimization problem is interesting as there is no polynomial solution to maximize the volume and minimize the weight at the same time. In this regard, the Knapsack problem is an NP-hard optimization problem. Another interesting part of the Knapsack problem is the constraint. An optimization method that can capture such a relationship will outperform others. As MIMIC can model probability distribution and eventually convey underlying relationships, the MIMIC performs best for this problem closely followed by GA. Such a unique property of MIMIC may have resulted in presenting a consistently better result than other algorithms that do not have the same features. Further analysis is provided below.

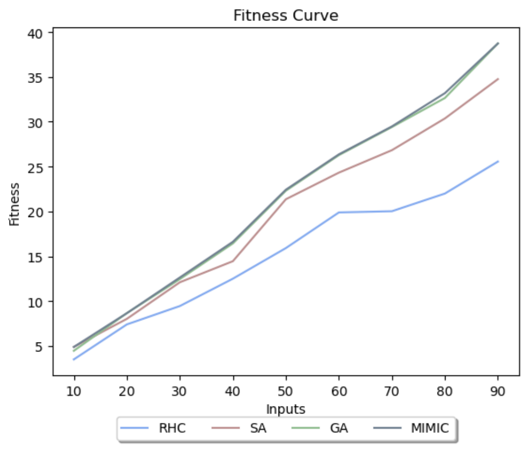

Fig. 13. Result Comparison

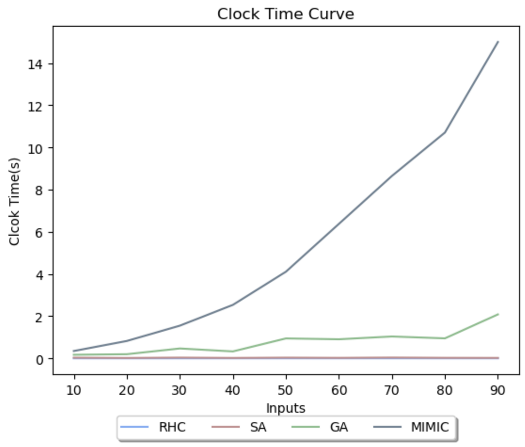

Fig. 14. Result Comparison

RHC
The RHC performed worst out of the other algorithms. The Knapsack optimization problem requires the optimization algorithm to exploit the structure of the problem to be able to produce the best solution. However, the SA does not remember its previous trials and only greedily moves upwards to the best neighbour. Figure 15 represents that changes in the number of restarts did not change much regarding the fitness score. This again highlights that an optimal solution cannot be easily reached by just simply starting at random points and hoping that it would get to solve the problem. Rather, an algorithm that overcomes forgetting the previous state can perform well on the Knapsack problem.

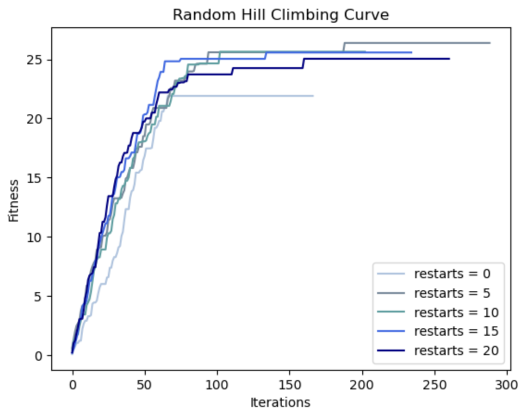

Fig. 15. RHC Hyperparameter Comparison

SA
The SA performed the second worst for the Knapsack problem. Similar to RHC, the SA does not have the property of remembering past behaviour. Capturing the constraints of the Knapsack problem to optimally adjust its future decisions is not viable with the SA. Hence, changing the temperature decay method does not improve the result significantly which can be seen in figure 16.

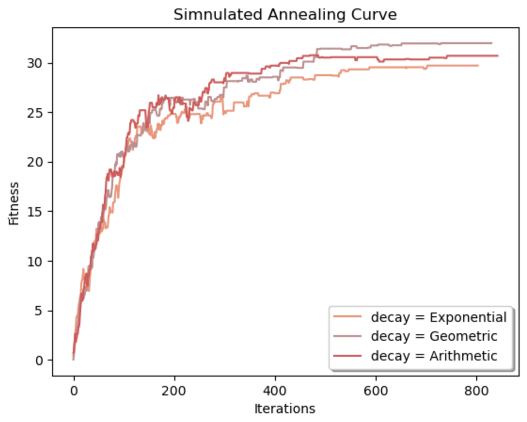

Fig. 16. SA Hyperparameter Comparison

GA
The GA performed second best after the MIMIC algorithm. The fact that GA uses exploration by implementing mutation and crossover and exploitation via selecting an elite population from the parent population, can help GA to find a good solution to the problem. Thus, GA continually performs better than algorithms that only explore but never exploit such as RHC and SA. Nevertheless, it does not explicitly remember the structure of the problem. As a result, GA performs slightly worse than MIMIC. Figure 17 shows that after 50 iterations, all hyperparameter combinations converged.

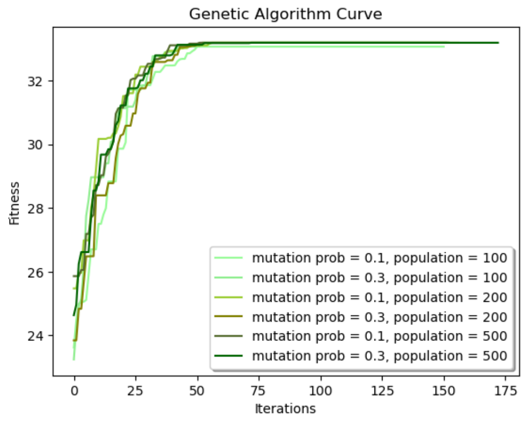

Fig. 17. GA Hyperparameter Comparison

MIMIC
The MIMIC benefits from the nature of the problem, which requires optimizing the problem given its constraint. The MIMIC retains the history of its decisions and learns the probability distribution of the data. The result, therefore, is the best out of four different algorithms. However, there still is room for further improvement by picking better hyperparameter values which can further increase the fitness score gap between MIMIC and GA. Figure 18 shows how MIMIC performs based on the keeping percentage and the population size. In previous optimization problems, other hyperparameter comparison plots of MIMIC represented that it is sensitive to its parameter values. Likewise, figure 18, again depicts some wide gaps in the fitness score between hyperparameter choices. With this example, keeping a percentage of 0.2 and a population size of 500 created the best performance. However, a more extensive hyperparameter search might find better hyperparameter configurations than the ones that are chosen for this trial.

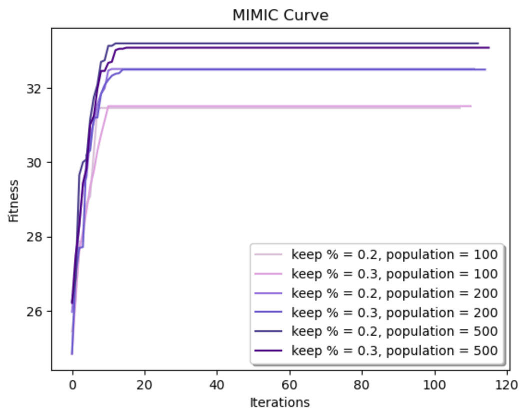

Fig. 18. MIMIC Hyperparameter Comparison

IV. Comparing with Neural Network

In this section, three optimization algorithms explored in this paper(RHC, SA, and GA) are used to find good weights for a neural network. The results are compared with the backpropagation on the same data used for assignment 1. The performance of different optimization algorithms is measured with accuracy and f1-score to give a fair judgement.

A. Optimization Methods

RHC
Figure 19 illustrates how a neural network with the RHC optimization method performs on Wisconsin breast cancer data. The right side of the plot in figure 19 shows the accuracy of the neural network with RHC. As the training set size increases, the accuracy also tends to increase on the test set. The trend applies the same with the f1-score. However, the overfitting seems to be occurring until the full dataset has been used. In other words, the performance on a training set is continuously performing better than the test set until the full dataset is used. As RHC randomly starts at any point of the search space, we can assume that the RHC might have found weights that are overly dependent on the training data. We can use some regularizers or increase the regularizing parameter to reduce the variance in this case to reduce overfitting.

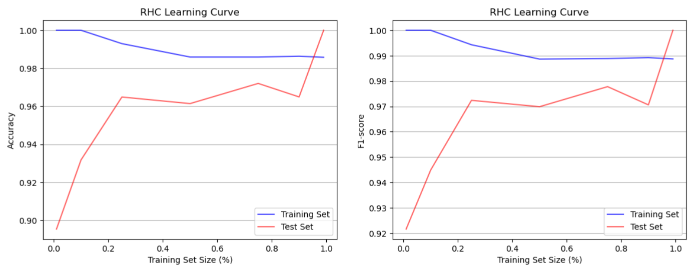

Fig. 19. RHC Learning Curve

SA
The SA reached a higher accuracy and f1-score compared to

RHC with less training set size as displayed in figure 20. Leveraging the temperature property to adjust the weights of the neural network could lead to finding better weights earlier in the training. However, the f1-score is lower than RHC. As f1-score is the harmonic mean of both classes, the SA might have optimized more towards one side of the class, which may result in a better score for one class over another. Interestingly, as more training size increases, the accuracy and f1-score are lower than RHC. The RHC might have benefited from its randomness to find a better result. Nonetheless, the overfitting issue is not as prominent as it is in RHC. The SA seems to have a relatively stable learning curve as the accuracy is roughly 95 percent for the test set and the gap between the training and test set is less than 2 percentage points. This shows that SA has a nice balance between bias and variance.

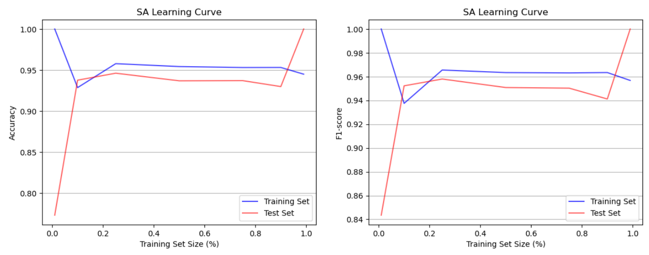

Fig. 20. SA Learning Curve

GA

The GA continuously perform better on this dataset compared to RHC and SA on all training set size ratio. The result applies also the same with the f1 measure. Again, this phenomenon could be explained by how GA operates. It explores and exploits the parent population to create elite offspring. Reflecting on the characteristics of the Wisconsin breast cancer data, finding the pattern of having breast cancer from the dataset can positively affect classifying breast cancer patients. The GA can find the most probable pattern for classifying breast cancer patients by exploitation. The GA can further sophisticate its decision boundary using exploration. Figure 21 depicts the learning curve of the neural network using GA. As the training set size increases the accuracy discrepancy gap between the training and test set narrows. More training example gives the GA more population size to work with, which is consistent with the analysis of previous optimization problems. GA showed improved performance during the hyperparameter tuning process when population size increases. Having more training examples allow GA to work on a more diverse pool of population, which helps GA to extract useful information from the parent population. As a result, it can also reduce the overfitting problem. The neural network with GA also shows very good performance, which shows that the bias problem is a matter of issue in this case.

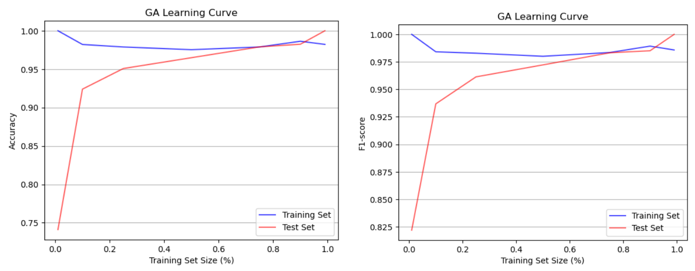

Fig. 21. GA Learning Curve

Backpropagation

The backpropagation adjusts the weight parameters of the neural network by taking the gradient of the error with respect to its weights. By taking the derivative, the backpropagation seeks to achieve the lowest gradient possible. When it cannot find the global optimum, it will at least converge to the best-fit approximation of the target function. This method has been proven to work surprisingly well with the neural network compared to other optimization techniques in solving many real-world cases. Figure 22 also highlights that both accuracy and f1-score are higher than the RHC, SA, and GA. Both accuracy and f1-score seem to produce a promising result. As the plots illustrate, the training set curve is high without any sign of underfitting. There is, however, some gap between the training and test set, which is a sign of overfitting. Nevertheless, the amount of difference between training accuracy and test accuracy is roughly less than 0.025 after the training set size of 20 percent. Hence, the overfitting problem should not matter too much. To improve the overfitting problem we can try to use a regularization technique for the neural network such as drop out which randomly drops some nodes during each training. We can also try to implement a weight decay method to penalise weight that is too big.

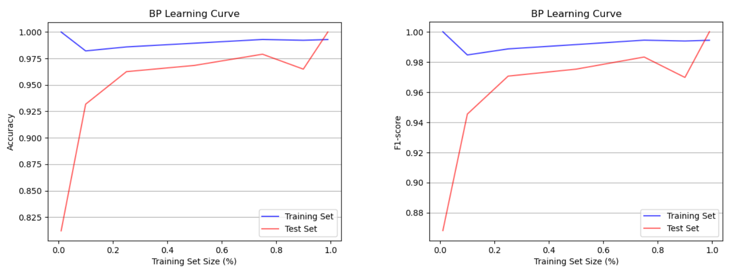

Fig. 22. Backpropagation Learning Curve

B. Results Comparison

The results of four different optimization algorithms are concisely summarized in figure 23. As described above, the best-performing optimization technique is backpropagation. The training, validation and test set accuracy are highest compared to other methods. The GA is the second-best optimization method for the Wisconsin breast cancer classification problem. The validation data score was almost the same as backpropagation. However, the test set result was noticeably lower than the backpropagation. The RHC performed slightly worse than GA and backpropagation on the training and validation dataset. The test set score, however, was comparably lower than those two algorithms. Lastly, the result of SA was significantly lower for all training, validation, and test scores. However, it took the least amount of fit time as portrayed in figure 24. Considering how fast SA is, it could be a reasonable alternative when the computational resource is limited. Nevertheless, the backpropagation also took quite a short time to reach the best-performing result. This concludes that backpropagation is the optimal method for this problem. Although RHC is known for its fast computation time, the number of restarts and the availability of parallel computing decides the computation time. The fact that RHC is continually searching for the best neighbour for n number of restarting times can increase the amount of time it takes to converge. Finally, the GA was quite computationally expensive as it has to first compute the fitness score of all the parent populations and then calculate the probability that the parent will be selected next stage. Once the candidates are selected with the probability, then it needs to operate the cross-over stage followed by the mutation step. The GA iterates this process until convergence, which can be time-intensive. The hyperparameter search spaces are provided in Appendix A.

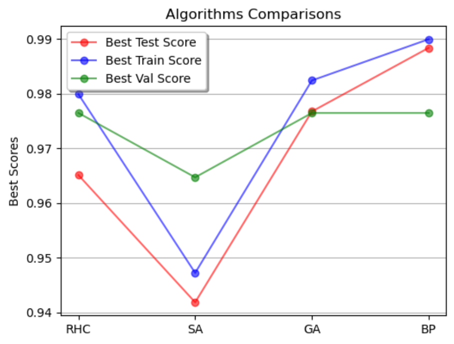

Fig. 23. Result Comparison

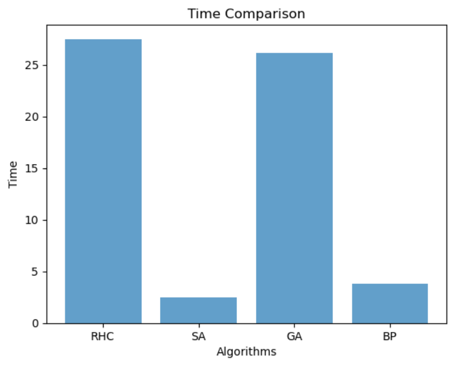

Fig. 24. Computation Comparison

The loss curve per iteration is presented in figure 25. The overall loss result shows that the neural network with backpropagation(labelled as NN) recorded the lowest loss on all iterations followed by the neural network using GA. It is interesting to see the SA and RHC loss curve compared to other models. As these two algorithms are based on randomness the loss curve is more wildly fluctuating before converging. On the other hand, GA shows a systematic staircase-like drop in the loss. As GA systematically goes through selection, crossover, and mutation steps to iteratively produce superior offspring the loss curve also decreases in such a manner. Lastly, the backpropagation corrects its weight based on the derivative of the error with respect to weights times some small learning rate that determines the step size. Thus, it smoothly adjusts the weights, which are also represented in the loss curve.

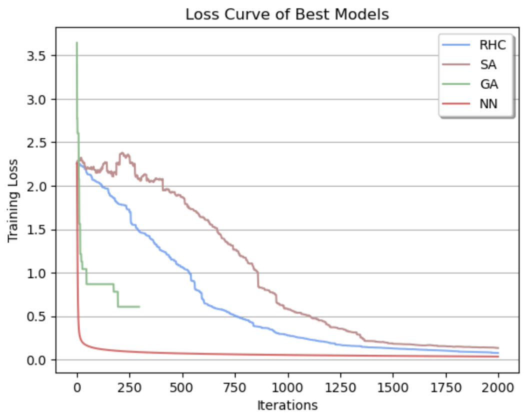

Fig. 25. Computation Comparison

V. Conclusion

In this analysis, four different optimization algorithms were first compared on contrasted on three different optimization problems. Depending on the type of problem, each algorithm showed its strengths. The second part of the report compared RHC, SA, GA, and backpropagation on the Wisconsin breast cancer data used in the previous assignment. The neural network paired with backpropagation resulted in the best performance. In summary, experiments conducted in this report proved the no-free-lunch theorem that there is not one optimal algorithm that works best in all problems[2].

VI. Appendix

A. Hyperparameter Search Space

TABLE I
HYPERPARAMETER SEARCH SPACE

Learning Rate	[0.00001, 0.0001, 0.01, 0.1]
Number of Restarts	[2, 5, 10, 15, 20]
Temperature Decay	[exponential, arithmetic, geometric]
Population Size	[50, 100, 150, 200, 250, 300, 350]]

References

[1] Russell, S.,   Norvig, P. (2020). Artificial Intelligence: A Modern Approach (Pearson Series in Artifical Intelligence) (4th ed.). Pearson.

[2] Ho, Y.-C Pepyne, David. (2001). Simple explanation of the no free lunch theorem of optimization. 5. 4409 - 4414 vol.5. 10.1109/.2001.980896.