This assignment consists of two parts: randomized optimization algorithms on optimization problems and neural network optimization. To successfully complete both parts, I utilized Python including the libraries mlrose, scikit learn, and matplotlib. The algorithms implemented for part two (neural network optimization) includes Randomized Hill Climbing (RHC), Simulated Annealing (SA), Gradient Descent (GD) and Genetic Algorithm (GA). Part one involves RHC, SA, GA and Mutual-Information-Maximizing Input Clustering (MIMIC) algorithms.

Part One: Randomized Optimization Algorithms

First I conducted hyperparameter tuning for the following three optimization problems: Four Peaks, Continuous Peaks, and One Max. The hyperparameter tuning was conducted on the RHC, SA, GA, and MIMIC algorithms. The hyperparameter for RHC is the number of restarts, which determines how many times the algorithm will restart its search for the optimal solution from a randomly generated initial state. These restarts help to minimize the risk of the solution getting stuck in a local optimum (when the algorithm converges). The hyperparameters tuned for SA are the initial temperature, temperature decay rate, and minimum temperature. The initial temperature refers to the starting point of the annealing process and helps to control the level of randomness in the search. The decay rate is a multiplier applied to the temperature at each step to reduce it and focus the search. The minimum temperature represents the termination (or convergence) criteria. The hyperparameters tuned for GA are population size and mutation probability. Population size is the number of solutions present in each generation of GA. The mutation probability is the likelihood that a particular feature will mutate by introducing small random changes that will help the algorithm escape the local optima. The hyperparameter tuned for MIMIC is the keep percentage parameter which represents the proportion of the samples to keep from one iteration to the next.

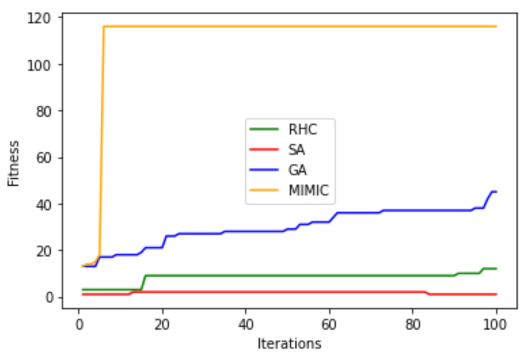
Algorithm Name:	**RHC**	**SA**	**SA**	**SA**	**GA**	**GA**	**MIMIC**
Hyperparameter:	# of Restarts	Initial Temperature	Decay Rate	Minimum Temperature	Population Size	Mutation Probability	Keep Percentage
Four Peaks	100	32	0.2	1	100	0.2	0.25
Continuous Peaks	75	8	0.8	0.1	200	0.2	0.75
One Max	75	8	0.8	0.1	100	0.2	0.25

For the algorithms implemented for each optimization problem, I used the optimal values listed above for the hyperparameters. Then, I calculated the time it took to run each algorithm and plotted the number of iterations vs fitness values. I also analyzed how different problem sizes (10, 50, and 100) impacted the fitness values for each algorithm.

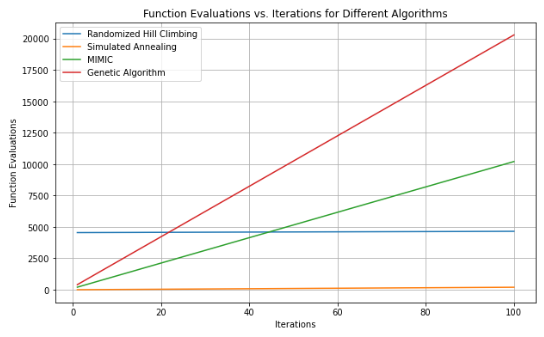
Four Peaks

The Four Peaks optimization problem is simple and optimal for testing algorithms such as RHC, GA, SA, and MIMIC. The fitness for this problem involves returning the maximum after counting the number of zeros at the start and the number of ones at the end. This results in two small peaks at high concentrations of zeroes and ones. If the number of ones and zeroes exceeds the threshold value, then the fitness function gets bonuses and results in two larger peaks. Together, this results in four peaks: two local maxima and 2 global maxima.

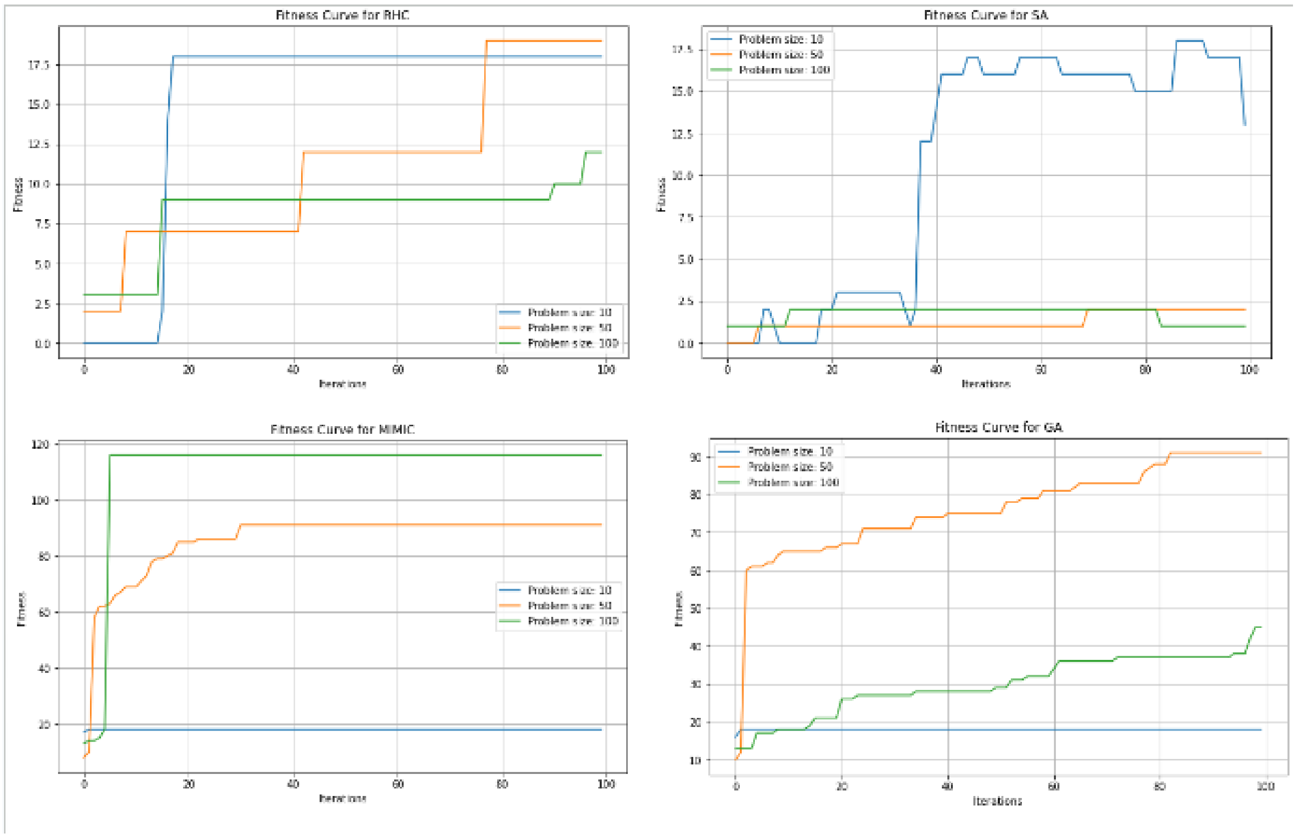
The plot below depicts the fitness curves for each algorithm against the number of iterations. MIMIC appears to have the highest and least fluctuating fitness. GA exhibits a lot of



fluctuations and never truly achieves convergence or stabilization of the fitness curve, which could be due to the probabilistic selection of the parents, crossovers, and mutations all of which can introduce variability. SA appears to have the lowest fitness, which could be attributed to the optimal hyperparameter values obtained earlier. For the Four Peaks problem, SA required the initial temperature to be 32. This is a high initial temperature, which may make it easier for the algorithm to accept worse solutions early on and slow down the convergence process.



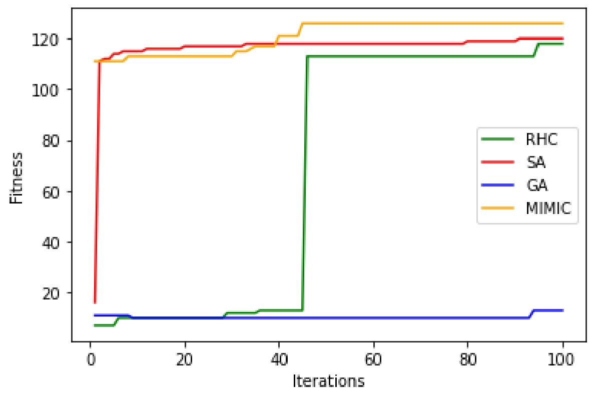
The plot on the left depicts the function evaluations across 0-100 iterations for each algorithm in the Four Peaks problem. Both the MIMIC and GA algorithms continuously increase over all the iterations, but never achieve a plateau or convergence. This indicates that the optimal solution has not been found despite continuously searching for it. This affirms the need for adjusting the parameters or increasing the number of iterations in order to observe convergence. GA does have the steepest curve, which means that it is the most efficient algorithm for the Four Peaks problem. RHC and SA both remain fairly constant with no increase or plateau. This stagnation indicates that it is unable to find a better solution. With RHC, it remains constant at close to 5000 so that suggests that RHC found convergence very early on.



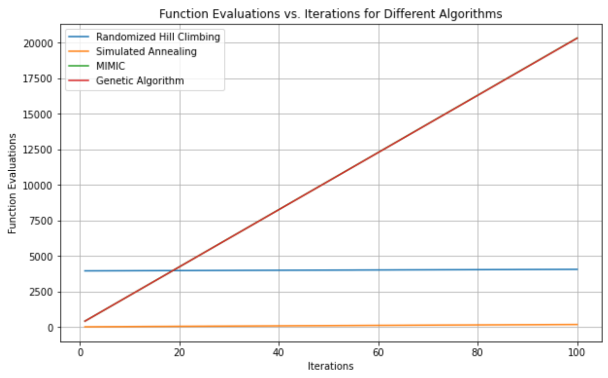
The four plots above depict the fitness curves for each problem size: 10, 50, and 100. RHC observes the highest fitness with the smallest problem size. Having a small problem size will narrow the search space, which may make it easier to find the optimal solution. SA again observes higher fitness with the smallest problem size. This may be due to the high initial temperature setting for SA in the Four Peaks problem. Larger problem sizes may be more sensitive to the initial temperature, thus making it difficult to achieve convergence. The largest problem size had the highest performance with MIMIC. While larger problem sizes have higher variability and complexity, the probabilistic modeling approach within MIMIC allows it to effectively deal with noise and achieve better performance. With GA, the smallest problem size appears to achieve convergence early on and have the lowest fitness. Having a small population can restrict the diversity of the population, thus limiting the exploration of the search space and achieving subpar solutions.

Continuous Peaks

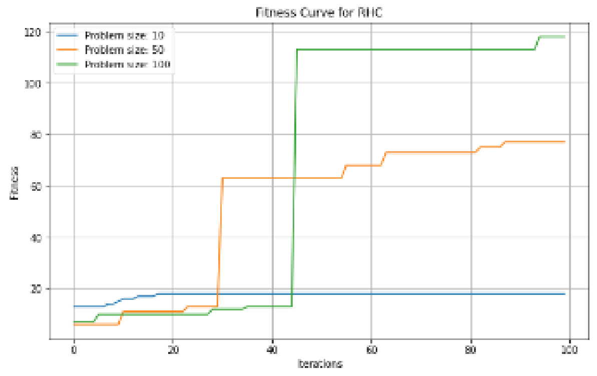
The Continuous Peaks optimization problem is similar to the Four Peaks problem. While Four Peaks restricts the ones and zeros to be at the ends of the solution string, the Continuous Peaks problem allows them to be present anywhere in the string. The Continuous Peaks problem is interesting because it can navigate complex fitness landscapes. Moreover, it has real-world applications in that it is able to pursue multimodal optimization.

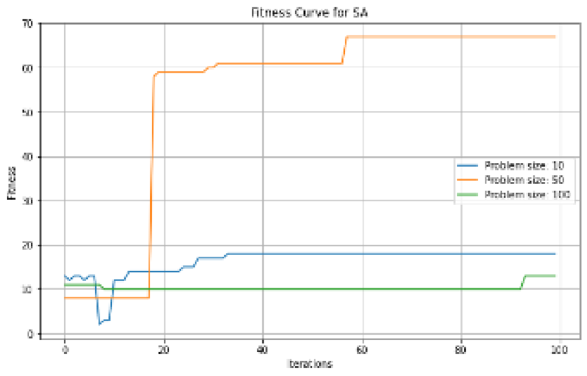


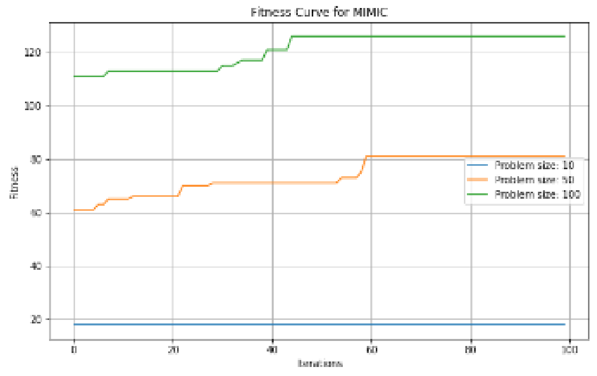
It is apparent from the plot on the left that RHC, SA, and GA eventually achieve convergence and exhibit high fitness overall. In contrast, MIMIC has low fitness regardless of the number of iterations. MIMIC generally relies on modeling dependencies between variables to guide the search. The Continuous Peaks problem contains a deceptive fitness landscape, which may make it difficult for the MIMIC algorithm to discern between promising and non-promising solutions.

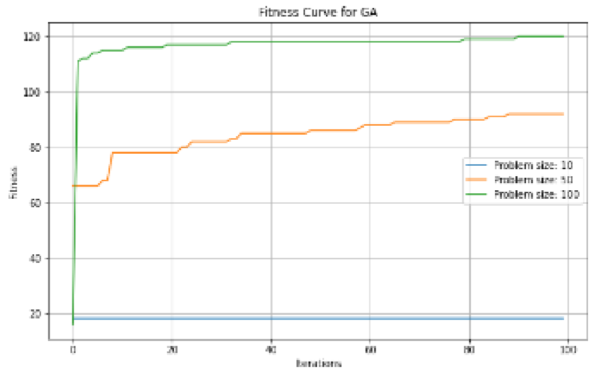


The function evaluations plot on the left shows that the GA algorithm continually increases steeply over the iterations but never achieves convergence within the hundred iterations. Meanwhile, MIMIC and SA algorithms remain constant. Again, this can be adjusted with parameters or increasing the range of iterations. The four plots below depict fitness curves for different problem sizes in the Continuous Peaks problem. The largest problem size of 100 performed the best for RHC, MIMIC, and GA but performed the worst for SA. The Continuous Peaks problem is generally deceptive, so having a large problem size can allow these algorithms to have a broader search space and achieve optimal solutions better than these algorithms with the smaller problem sizes. The largest problem size performed the worst for SA, possibly due to the hyperparameter tuning as SA has a high decay rate of 0.8 which may heighten the risk of premature convergence.



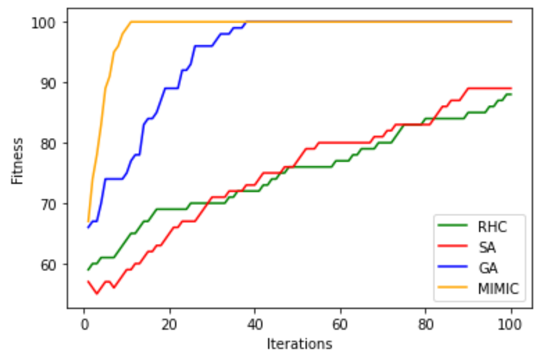




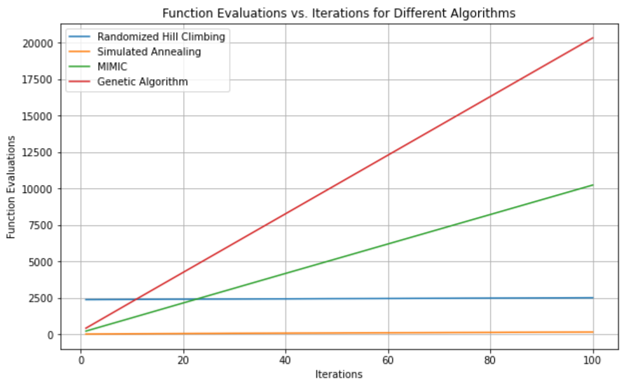


One Max

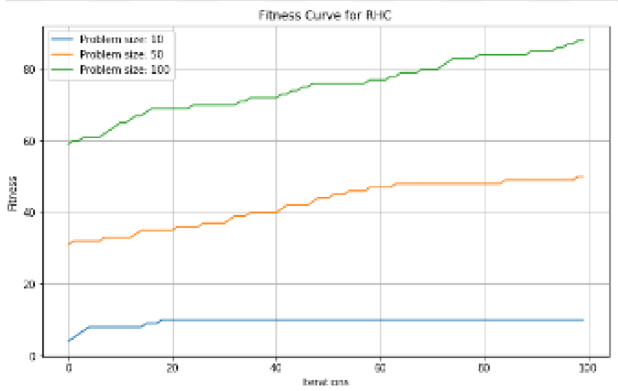
The One Max optimization problem is rather simple in that the fitness is the number of ones in the bit string. This ensures that there is only one optimum. Since this problem is represented in binary form, it can be applied to many real-world optimization problems.

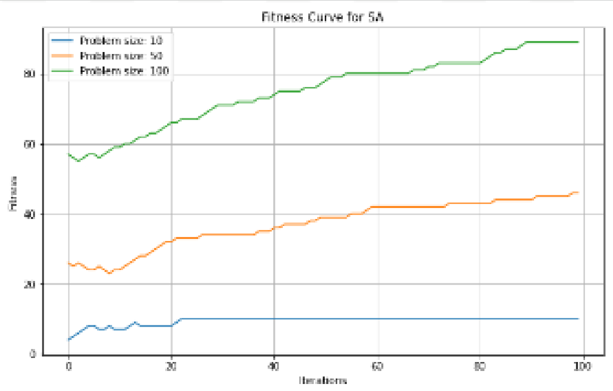


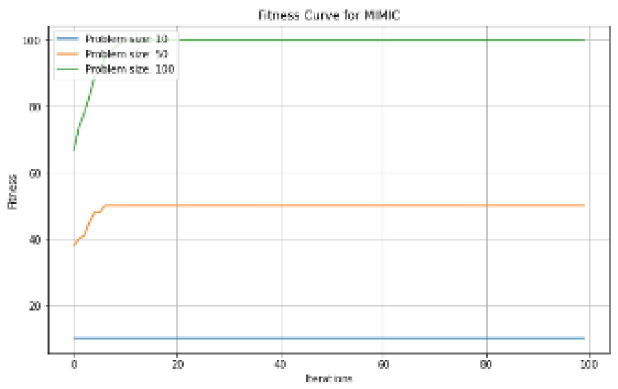
The plot on the left here depicts the fitness curves vs iterations for each algorithm in the One Max problem. MIMIC and GA observe the highest fitness. MIMIC converges earlier at around 10 iterations, while GA experiences far more fluctuations before converging at around 40 iterations. GA may experience a lot of fluctuations due to the design of this algorithm as the probabilistic selection can introduce more variability in the fitness. SA and RHC experience a lot of fluctuations as well and generally achieve lower fitness than MIMIC and GA. SA has a high decay rate of 0.8, which may increase premature convergence and increase the chance of achieving poor solutions. This may explain why SA has a lower fitness. RHC is prone to get stuck in the local optima, which may hinder the process of finding the optimal solution and result in lower fitness.

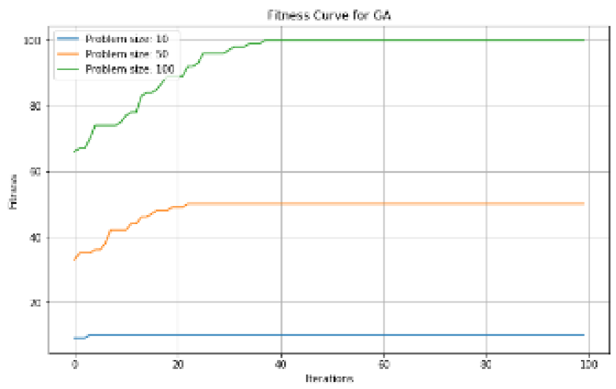


The function evaluation plot on the left is similar to the function evaluation plot for the Four Peaks problem in that both GA and MIMIC are continually increasing across the iterations while the other two algorithms remain constant. The design of these algorithms may explain the differences in function evaluations for these algorithms. GA and MIMIC both maintain multiple possible solutions simultaneously within their populations. RHC and SA, however, work with a single solution at a time. Since GA and MIMIC are able to handle several solutions at once, they tend to be more efficient in exploring the search space. However, none of these algorithms across all three optimization problems achieve convergence. This may occur due to the limited range of iterations or the improper adjustment of the hyperparameters. The four plots below depict fitness curves of problem sizes in One Max.









Interestingly, all four algorithms exhibit similar impact from the different problem sizes in the One Max problem. The largest problem size of 100 has the highest fitness, while the smallest problem size of 10 has the lowest fitness. The smallest problem size also achieves convergence earlier on than the other two sizes for all four algorithms. A larger problem size results in a longer bit string for the One Max problem. This increase in space can ensure that the risk of encountering local optima and achieving subpar solutions will decrease. The fitness curve plot for the MIMIC algorithm has far less fluctuations than the other three algorithms. The One Max problem is relatively simple as it is trying to maximize the number of ones in a string. The MIMIC algorithm utilizes a probabilistic model to capture the dependencies between bits or variables. Since the One Max problem is trying to maximize the number of ones, the MIMIC algorithm can more easily reflect the dominance of ones. This may explain why the fitness curves for MIMIC across all three problem sizes appear to be smoother.

Time to Run Each Algorithm for Each Optimization Problem (seconds)

	RHC	SA	GA	MIMIC	Average
Four Peaks	0.3198	0.01651	1.67524	1193.19997	298.80288
Continuous Peaks	1.2644	0.02427	6.51139	2107.20949	528.7523875

One Max	0.41758	0.008	1.64011	1397.58443	349.91253
Average	0.66726	0.01626	3.27558	1565.997963	

The table above captures the time in seconds it takes to run each of the four algorithms on the optimization problems: Four Peaks, Continuous Peaks, and One Max. On average, SA observes the fastest run time across all three optimization problems while MIMIC is the slowest. It is no surprise that the MIMIC algorithm runs the slowest as it utilizes iterative steps to refine its probabilistic model to represent the dependencies in the problem. SA combines global search and local search, which may make the algorithm run faster than the others. The average time for running all four algorithms is fairly similar among the three optimization problems.

For this section, I conducted hyperparameter tuning to obtain the optimal values for each algorithm in each optimization problem. I implemented these values for the algorithms and plotted the fitness curves. I also examined the impact of problem size on the fitness of these algorithms. For One Max and Four Peaks problems, the MIMIC algorithm observed the highest fitness while SA had the lowest. While MIMIC enjoys high fitness for these problems, it does have the slowest run time which is an important tradeoff to consider. SA may have observed low fitness for these problems due to the hyperparameter tuning. Four Peaks had a high initial temperature of 32, and One Max had a high decay rate of 0.8. Although these values were obtained with hyperparameter tuning, it may be better to attempt different parameter values to see if SA can perform better.

MIMIC observed low fitness for the Continuous Peaks problem, which may have occurred due to the deceptive nature of this problem. One Max and Four Peaks problems are rather straightforward and simple, while the Continuous Peaks problem has fitness plateaus. In the Continuous Peaks problem, the largest problem size performed well for all the algorithms except for SA. This could be due to the high decay rate tuned for SA in the Continuous Peaks problem, as that can risk premature convergence and lower fitness overall. Again, it might be valuable to explore different decay rates for this algorithm to see if the fitness can improve. When examining the function evaluations of each algorithm, it is clear that none of the algorithms achieved optimal solutions (or convergence). To address this issue, I would try adjusting the hyperparameters or I would expand the number of iterations to be beyond 100 iterations. To observe the impact of problem size on the different algorithms in these three optimization problems, I examined sizes of 10, 50, and 100. For further exploration, I would expand to much larger sizes to see if there is eventually a plateau at some unknown large problem size.

Part Two: Neural Network Optimization

The dataset is the Wisconsin breast cancer dataset with 30 numeric predictor variables and 1 target variable. The target variable indicated whether the breast mass is benign or malignant. The dataset came with the target variable encoded so I didn't have to encode it myself. To check for the balance of the dataset, I examined the distribution of the target variable output. Benign tumors made up 37.26% of the data, while malignant tumors made up 62.74%. I also calculated the Gini index to be 0.468, which affirms that this dataset is imbalanced. I split the dataset into a training and testing set. Then, I standardized the dataset so that it would be adequately scaled. In the previous assignment, finding the optimal hyperparameter values for the NN model was done in 15.48 seconds. The optimal value for hidden layer size is 1, while the optimal learning rate is 0.01. The accuracy for the model is 97.8% after hyperparameter tuning.

For this section of the assignment, I utilized these optimal parameters from assignment 1 to determine the amount of time to fit the training and test data when using ReLU activation vs sigmoid function activation for the neural network. I also evaluated the accuracy, precision, and F1 score for these two models. Then, I calculated the time it takes to fit the training data for each algorithm: RHC, GA, SA, and GD. I evaluated the fitness values for each algorithm with plots and calculated the accuracy values for predicting the test datasets. I utilized this approach for both ReLU and sigmoid function activation.

When implementing the neural networks with the parameters from assignment 1 and without optimization algorithms, I examined how the NN fared under ReLU vs sigmoid function activation. The Rectified Linear Unit (ReLU) activation returns zeros for negative x input and the input itself for any positive input. The sigmoid function activation is also known as the logistic function, while the ReLU function is a piecewise linear function. The sigmoid function is often suitable for binary classification problems as it converts the input to a binary 0, 1 range.

Activation	**Train Fit Time**	**Test Fit Time**	**Accuracy**
ReLU	0.324	0.002	0.965
Sigmoid	0.191	0.0003	0.956

The table above outlines the time it took for the neural network to fit the training data and the test data. The table also includes the accuracy scores. It is apparent that the neural network models are much faster at fitting the testing data than the training data, which can be attributed to how the test set is much smaller than the train set. Between the two activation functions, they appear to have similar and high accuracy scores at around 96%. The sigmoid activation function is faster than the ReLU function. I was surprised to see this because ReLU is generally known for being more computationally efficient than sigmoid functions. Since the sigmoid function does incorporate binary data, it makes sense that it was more efficient than ReLU for the breast cancer dataset as this dataset has a binary target variable.

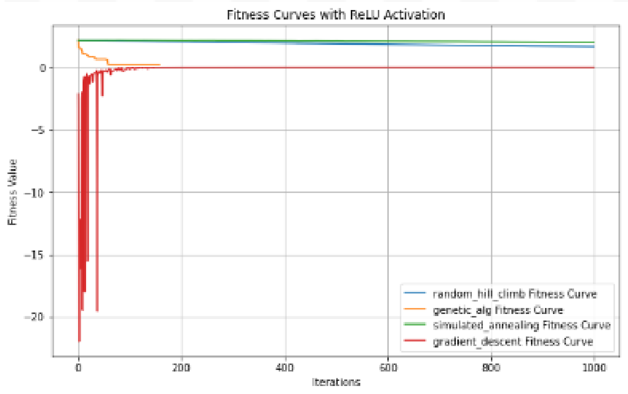
The table below exhibits the results for the train fit time and the accuracy in predicting the test dataset when running each optimization algorithm for neural network models with the ReLU activation functions.

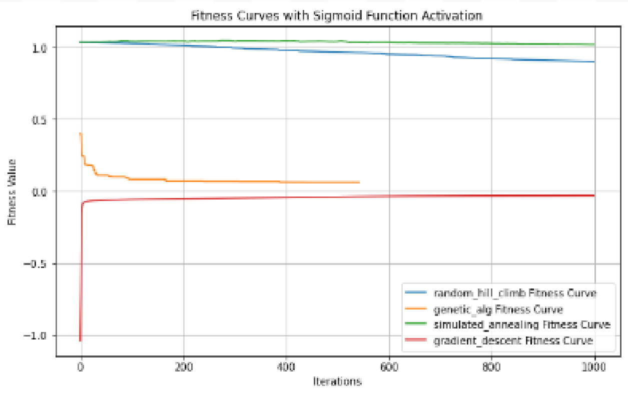
NN Models with ReLU Activation Function				
Algorithm	**RHC**	**GA**	**SA**	**GD**
Train Fit Time	3.1	78.35	6.53	6.47
Accuracy	0.49	0.95	0.43	0.94

The table below exhibits the results for the train fit time and the accuracy in predicting the test dataset when running each optimization algorithm for neural network models with the Sigmoid activation functions.

NN Models with Sigmoid Activation Function				
Algorithm	**RHC**	**GA**	**SA**	**GD**
Train Fit Time	3.25	334.36	5.25	5.81
Accuracy	0.47	0.97	0.42	0.96

Interestingly, each algorithm had similar accuracy scores regardless of the activation function used in the model. For instance, the SA algorithm exhibited 43% accuracy with ReLU and 42% accuracy with sigmoid. It is evident that the activation function utilized did not have a profound impact on accuracy. However, it did impact how long it took to fit the training data. This makes sense as ReLU is linear in nature, while sigmoid is logistical. GA with sigmoid activation was much slower than GA with ReLU activation. RHC with sigmoid activation was slightly faster than ReLU, while both SA and GD with sigmoid activation were a little slower than ReLU activation. The models with sigmoid activation were mostly slower than the models with ReLU activation, which does contradict the pattern observed in the models earlier without the optimization algorithms. Sigmoid activation functions are known for being less computationally efficient than ReLU activation functions, so these results make sense in





reflecting this.

The fitness curve plot on the left is for the models with ReLU activation. It is apparent that the RHC and SA models observe similar trends in that they have the highest fitness values and observe a slight decrease as iterations progress. This decrease is fairly stable, which indicates that convergence is occurring. GA and GD algorithms began to observe convergence (stabilization of the fitness values) at around 70 iterations. Up until this point though, the GD algorithm experiences many fluctuations in the fitness curve. The GD algorithm utilizes a deterministic approach, while the other three algorithms have randomized approaches. Randomized approaches make it easier to escape local minima and explore a broader solution space, which may explain why RHC, SA, and GA observed far less fluctuations in fitness than GD. The fitness curve plot above on the right is for the models with the sigmoid function activation. All four algorithms observed similar fitness curves but with much smoother curves and minimal fluctuations. While sigmoid functions generally result in slower computation, it does observe slower convergence. This might help us understand why there are far less fluctuations in the fitness for the fitness curve plot of the sigmoid functions.

Without the optimization algorithms, both ReLU and sigmoid NN are much faster than the models that include the optimization algorithms. Despite being much faster, they end up with high accuracy scores. GA and GD observe high accuracy scores, while RHC and SA do not. This is interesting as GA and GD generally observed more fluctuations in the fitness curve. While these fluctuations are generally seen in a negative light, they might be signifying that GA and GD are more sensitive to the hyperparameters and the data. This heightened sensitivity to learning the data may result in more fluctuations and high accuracy scores, especially in comparison to RHC and SA.

Works Cited

Hayes, G. (2019, November 3). mlrose Documentation Release 1.3.0.

Isbell, C. L. (1996). *Randomized Local Search as Successive Estimation of Probability Densities* (thesis).

LaetitiaVanCauwenberge. (2018, November 10). *Deep Learning Networks: Advantages of ReLU over sigmoid function*. Data Science Central. https://www.datasciencecentral.com/deep-learning-advantages-of-relu-over-sigmoid-function-in-deep/#:~:text=Relu%20%3A%20More%20computationally%20efficient%20to,better%20convergence%20performance%20than%20sigmoid.

Spears, W. M. (1993). Crossover or mutation? *Foundations of Genetic Algorithms*, 221–237. https://doi.org/10.1016/b978-0-08-094832-4.50020-9

Tutorial - machine learning weight optimization problems¶. Tutorial - Machine Learning Weight Optimization Problems - mlrose 1.3.0 documentation. (n.d.). https://mlrose.readthedocs.io/en/stable/source/tutorial3.html#solving-machine-learning-weight-optimization-problems-with-mlrose

University of Vienna. (n.d.). Evolutionary Learning Unit 7. Vienna.