# Assignment 2

Joe Kraemer

## I. Randomized Optimization of Discrete Problems

### A. Methods

For every problem and every algorithm, a convergence plot is generated by running the algorithm 10 separate times each on a new problem and initial state. With each new problem, we are also creating new initial states and random seeds. This will expose variance that could make one problem harder as well as the variance from one initial state being more difficult than another. The mean of these runs is then plotted along with the variance as the shaded section.

To find the optimal hyper parameters for each algorithm, a grid search was used to expose interplay between hyper-parameters. For example, with MIMIC the population size vs keep percentage. These two values need to be scaled together.

When creating the validation curves for each hyper-parameter, the algorithm was tested 7 times at each hyper-parameter value. Similar to the other repeated tests, for each test, a new problem and initial state where created. The curve represents the mean and the shaded area represents one standard deviation of the data.

### B. Convergence Criteria

To select convergence criteria, one must look at an fitness vs iteration plot. This will show how the algorithm behaves when going to infinity. For all of my algorithms, I will use a $\delta$ convergence method. This method tracks how many attempts have been made to increase the fitness function by some $\delta$. After some number of attempts *a*, with no increase of $\delta$, the function is declared as converged. Since we are working with discrete problems, I set $\delta = 0$.

For Random Hill Climbing with Restarts (RHC) setting the convergence criteria is based on the dimensionality of the problem. Since with RHC, the algorithm checks a random neighbor to see if it has a higher fitness score than the current location. So to ensure full convergence, we want to set the max_attempts to a number that will allow the algorithm enough attempts to visit every neighbor at least once. The odd of us picking a unique neighbor first is $p = \frac{n}{n}$ (where $n$ is the number of neighbors). The odds of us picking a unique neighbor for the second attempt would be $p = \frac{n-1}{n}$. This generalizes to $p = \frac{n-k}{n}$ (where $k$ is the $k$th attempt) which is a geometric distribution. The expectation of a geometric distribution can be generalized to $E(n) = \sum_{k=1}^{n} \frac{n}{k}$. Using this function we can get an mean number of attempts to visit each neighbor once for every problem dimension. If we want to have a 99.7% confidence that this will happen, then we simply add 3 $\sigma$ of variance to our expectation of E(n). The variance of the sum is the sum of the variances. The equation for variance of a geometric distribution is $\sigma^2 = \frac{1-p}{p^2}$. Given our probabilities $p = \frac{n-k}{n}$, we can create a general equation $\sigma^2 = \sum_{k=1}^{n} \frac{1-\frac{n-k}{n}}{(\frac{n-k}{n})^2}$. Using this equation for variance and the previous equation for mean expectation we get the equation below.

$$MaxAttempts(n) = E(n)+3*\sigma = \sum_{k=1}^{n} \frac{n}{k}+3\sum_{k=1}^{n} \sqrt{\frac{1-\frac{n-k}{n}}{(\frac{n-k}{n})^2}} \tag{1}$$

So for example, if our problem complexity $d = 30$, then the number of neighbors would be $n = 2 * d = 30$ because we are working in discrete optimization spaces and therefore max_attempts would be 505. This equation will be applied in all subsequent problem spaces as this criteria is independent of the underlying fitness function.

As SA performs a similar method of moving about the problem space, applying this same convergence criteria makes sense. This ensures that SA has evaluated its entire immediate surroundings and we can confidently say that the function has converged and there is no more room for improvement. Therefore, I will use this same $\delta$ convergence method using max_attempts to control the completion.

I will apply the same $\delta$ convergence method to MIMIC and GA, but the way that they converge is not as easy to mathematically model as this is dependent on the specific problem space and the hyperparameters of each algorithm. Given this, I will discuss these later with each problem.

### C. OneMax - Simulated Annealing

OneMax is a simple fitness problem that sums the number of bit that are 1 in a vector. This means that the problem space has one global optima and its basis of attraction spans the entire problem space. With a problem like this, we expect that RHC and SA should achieve the global maximum because there are no local optima for them to get stuck at. We would also expect that MIMIC and GA will also achieve the global maximum, but the main difference will be in function evaluations and wall clock time. Given the additional overhead of MIMIC and GA, I expect these algorithms to take significantly more time.

In Figure 1, we can see that all of the algorithms are able converge to the global optima at all problem complexities. This is expected because there are no local optima for the algorithms to get stuck at, so given enough iterations, they will all make it to the global optima.

Looking at the convergence plot (Figure 2), we can see that GA and MIMIC were able to converge in less iterations, that does not tell the full story. Looking at the SA curve, we can see that there is quite a bit of variance in the middle portion. This is because of the way SA performs random jumps. Early on, all of the randomly sampled points were had a higher fitness than its current position so the jump was made automatically. As the algorithm rose to a higher

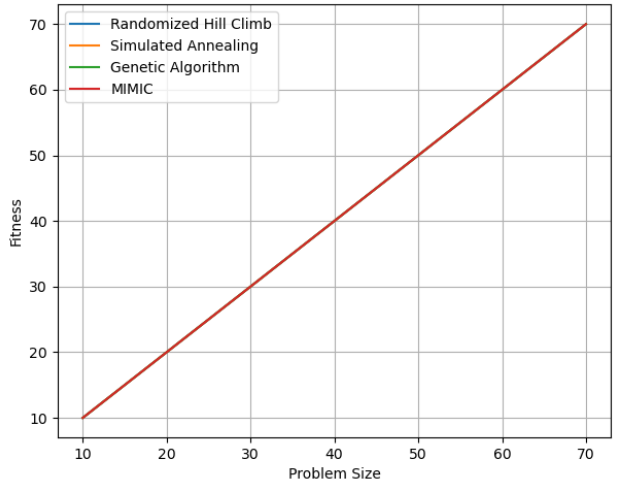


Fig. 1: Plot of Fitness vs Complexity of Various Algorithms, on OneMax.

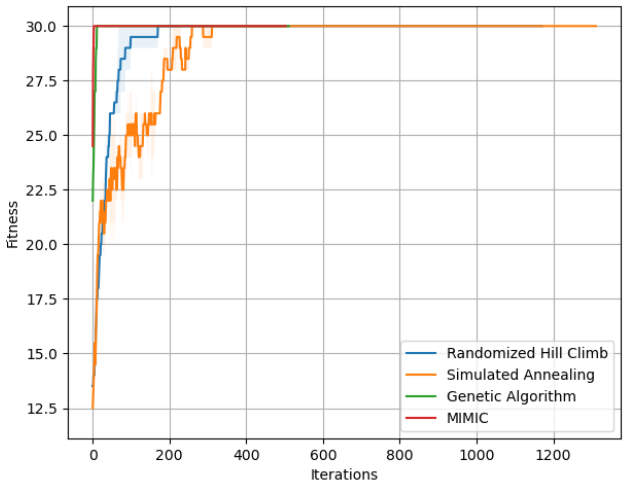


Fig. 2: Plot of Fitness vs Iterations of Various Algorithms, on OneMax (this plot was cropped on the x axis to highlight variance in SA).

fitness, the randomly sampled points could have been lower than its current position because the temperature value was still high and permitted this. This behavior is seen in the graph where SA appears to plateau and has a lot of variance as different runs were able to move faster than others. As the graph progresses, we see that the variance reduces. This is because the temperature is also reducing which restricts the size of backward jumps that can be made. We can see that GA and MIMIC were able to converge in less iterations, that does not tell the full story. But these algorithms are more computationally expensive and use many more function evaluations per iteration. Looking at this convergence plot, we can confirm that our delta convergence was set correctly with the previously discussed equation as RHC and SA are both successfully converging with 100% success. For this problem the complexity was $d = 50$ which means a max_attempts=896. We can confirm that no function improvements were made within that range. For MIMIC and GA, we can see that in all of the runs, they are converging much faster than RHC and SA. It would be a very safe assumption here to divide the max_attempts value returned from Equation 1 in half for these two algorithms. This gives us a safe margin of error to guarantee convergence.

In Figure 3, we can see that SA and RHC have a much faster computation time than GA and MIMIC. Not only that, but wall clock time for GA and especially MIMIC are scaling much faster with problem complexity than GA and RHC. This is because GA and MIMIC are making many more function calls than SA and RHC.

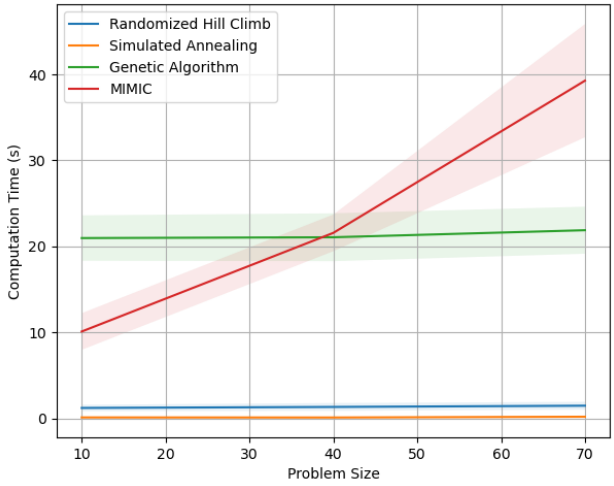


Fig. 3: Plot of Computation Time vs Problem Size of Various Algorithms, on OneMax.

Looking at Table I, we can see that GA and MIMIC are making orders of magnitudes more function evaluations. If this function evaluation was more difficult, then the time cost of running GA and MIMIC would be orders of magnitude higher than SA and GA. This shows that with problems without local optima, SA is a much better algorithm choice because it has much better time complexity and will still achieve the global optima.

| Algorithm | Total Fxn Evals | Convergence Time (sec) | Time / Iterations (ms) |
|---|---|---|---|
| RHC | 1144 | 0.58 | 0.51 |
| SA | 1325 | 0.05 | 0.04 |
| GA | 102616 | 11.11 | 21.72 |
| MIMIC | 101106 | 11.02 | 21.87 |

TABLE I: Table of Function Evaluations Statistics for Various Algorithms on One Max.

We can get a better look at the SA jumping behavior exhibited in Figure 2 by looking at Figure 4. We can see that Arithmetic decay is clearly the slower decay method by observing the high variance throughout the plot. With this decay method, the temperature takes longer to decay, so the algorithm will make random jumps to points with lower fitness. On a problem like OneMax where there are no local optima, there is no reason to explore, so the best choice here is to pick the fastest decay method, Exponential so that Simulated

Annealing only takes steps towards the global optima sooner. Selecting this decay method is obviously biasing the algorithm towards functions with no local optima, but in doing so, we reduce the variance as the algorithm will start to move towards the optima in less iterations. This also puts a lot of bias on the initial state, but again because we know that there is no local optima, it is okay to add this kind of inductive bias.

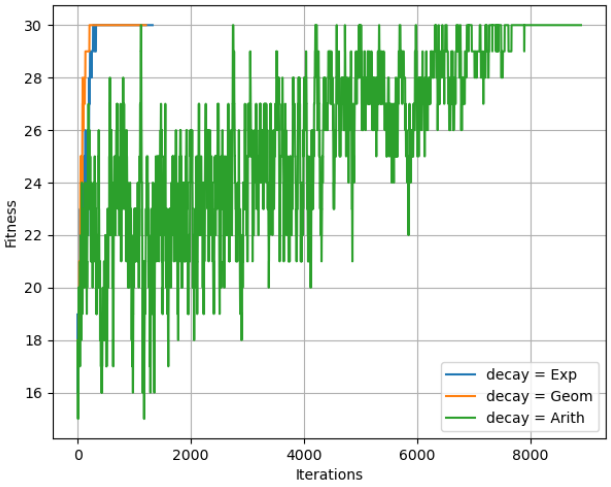


Fig. 4: Plot of Fitness vs Iterations of decay methods for Simulated Annealing, on OneMax.

| Restarts | Total Fxn Evals | Run Time (sec) | Time / Iterations (ms) | Fxn Evals / Iteration | Avg Fitness | Fitness Std |
|---|---|---|---|---|---|---|
| 0 | 1113 | 0.07 | 0.06 | 1.0 | 30.00 | 0.00 |
| 4 | 1097 | 0.32 | 0.30 | 1.0 | 30.00 | 0.00 |
| 8 | 1094 | 0.51 | 0.47 | 1.0 | 30.00 | 0.00 |
| 12 | 1109 | 0.73 | 0.66 | 1.0 | 30.00 | 0.00 |
| 16 | 1106 | 0.91 | 0.83 | 1.0 | 30.00 | 0.00 |

TABLE II: Table of Random Hill Climbing with various restart values on OneMax.

Looking at the Table II, we can see that increasing the number of restarts has little affect on the overall number of function evaluations and has a negative relationship with the overall run time. This is because its better to just continue to optimize instead of restarting from a place that is lower down the curve.

| Pop Size | Total Fxn Evals | Run Time (sec) | Time / Iterations (ms) | Fxn Evals / Iteration | Avg Fitness | Fitness Std |
|---|---|---|---|---|---|---|
| 5 | 7775 | 1.29 | 1.00 | 6.0 | 30.0 | 0.0 |
| 30 | 31574 | 4.89 | 4.80 | 31.0 | 30.0 | 0.0 |
| 50 | 51734 | 7.83 | 7.71 | 51.0 | 30.0 | 0.00 |
| 100 | 102275 | 14.58 | 14.39 | 100.9 | 30.0 | 0.0 |

TABLE III: Table of Genetic Algorithm with various population sizes on OneMax.

In Table III, we can see that reducing the population size has a huge benefit on the total function evaluations and overall run time, but still achieves perfect convergence with every value. Because any slightly positive move is the correct direction, its better to halve the population more frequently. With larger population sizes, we end up evaluation too many low fitness samples. With lower populations we are relying on the mutations to move us in the positive direction. While decreasing population is effective on this specific problem, it reduces the generality of GA and adds inductive bias.

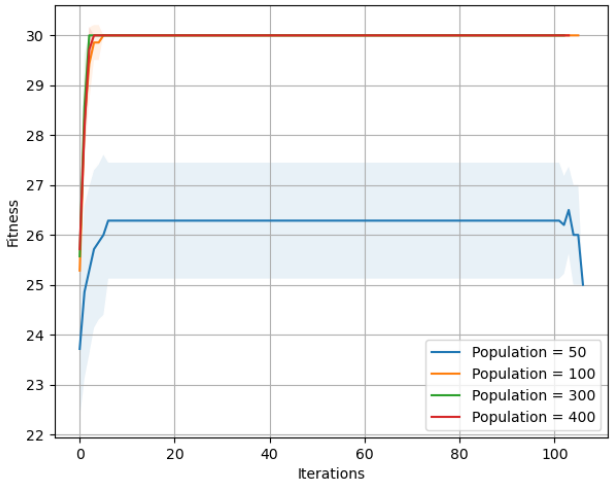


Fig. 5: Plot of Fitness vs Iterations of different population sizes for MIMIC, on OneMax.

In Figure 5, we can see that reduction in population size can reduce the effectiveness of MIMIC achieving the optima. If the population does not cover the problem space enough, then MIMIC is not able to properly approximate the Probability distribution of problem.

### *D. 4-Peaks - Genetic Algorithms*

Four peaks is an optimization problem where fitness is awarded by the number of 0s at the beginning of the bit string and number of 1s at the end of the bit string. There is a large bonus (equal to the dimensionality of the problem space) that is awarded if the number of 0s at the beginning and 1s at the end are both larger than some threshold. There are two local optima with wide basins of attraction where there are either lots of 0s at the beginning or lots of 1s at the end. There are two large global optima at the edge of the problem space where the bonus is awarded. These local optima attraction basins will likely catch RHC and SA which are only looking at their neighbors and will tend to just increase the 0s at the beginning or the 1s at the end. In a problem like this, we would expect that optimization algorithms that capture relationships between the features to excel. Specifically we should expect GA and MIMIC to achieve high fitness scores. I expect that GA should be able to converge faster than MIMIC.

As we can see in Figure 6, GA, RHC, and SA all seem to scale linearly with the increase in problem complexity while MIMIC appears to grow quadratically. This is because MIMIC is analyzing many potential structures while GA is inherently representing the problem structure. More explicitly, GA makes the assumption that separate dimensions of the problem space

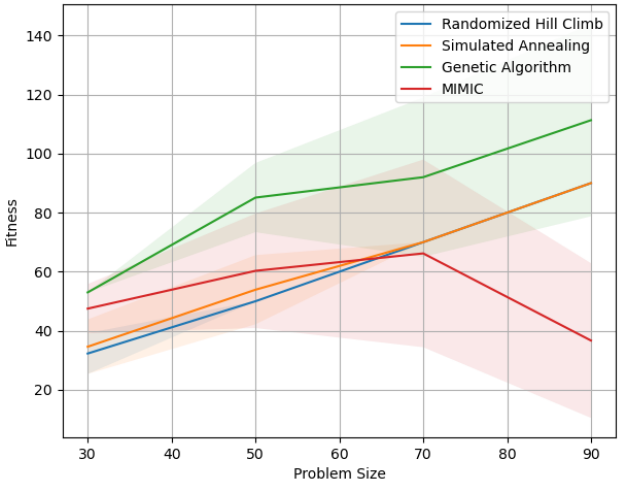


Fig. 6: Plot of Fitness vs Complexity of Various Algorithms, on FourPeaks.

contribute to the overall fitness value. While this inductive bias might limit the effectiveness of this algorithm on a broad range of problems, it just so happens that Four Peaks represents a structural relationship within this assumption. MIMIC doesn't prescribe a specific structure to the relationships between features and so it must analyze many more potential structures. This means that MIMIC will require a longer wall clock time to converge.

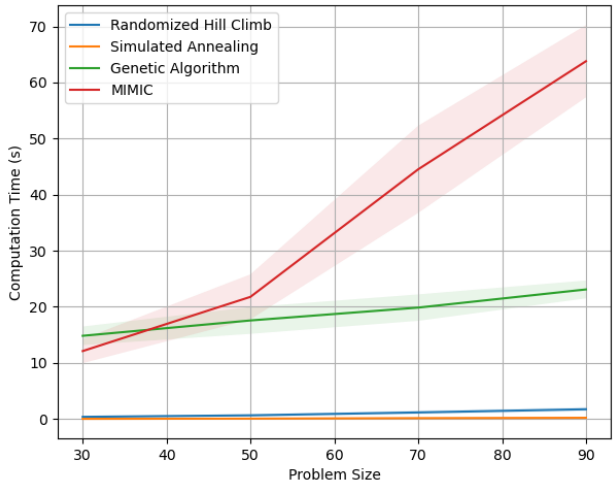


Fig. 7: Plot of Computation Time vs Complexity of Various Algorithms, on FourPeaks.

Looking at Figure 7, we clearly can see that GA, SA, and RHC computation times all scale linearly while the MIMIC is scaling faster and perhaps geometrically. As the problem space increases, calculating the probability distribution become increasingly difficult. GA only increases linearly with problem size yet still achieves the same, if not better results over MIMIC. This is partially because MIMIC is calculating probability distributions and constructing the best dependency tree. Although the wall clock times might differ, GA has a higher total number of function evaluations than MIMIC (Table VII). This is interesting, but highlights the additional computation cost of approximation the probability distribution of the problem space using dependency trees and not just the computation cost of the function evaluations.

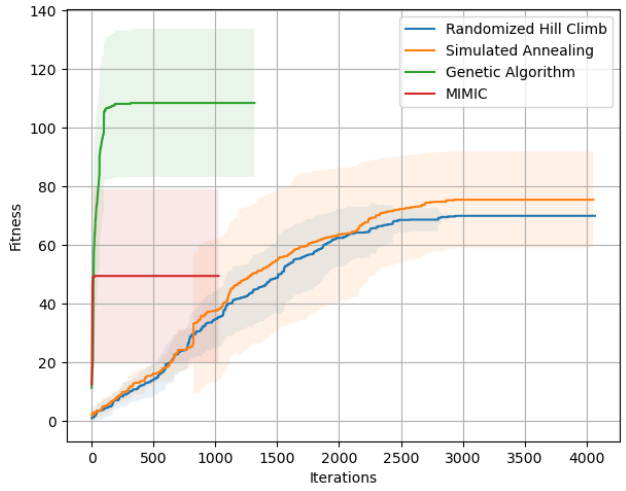


Fig. 8: Plot of Fitness vs Iterations of Various Algorithms, on FourPeaks of problem length 60.

For this problem the complexity was $d = 50$ which means a max_attempts=896. Looking at Figure 8 we can confirm that our convergence criteria was set correctly as there is no variance in RHC and SA as they reach the local optima. Looking at GA and MIMIC, they have some variance in their convergence plots but this is due to getting stuck in local optima, not a prematurely halted algorithm that still had the potential to improve.

In Figure 8, RHC and SA both achieve the local optima, but never are able to find the global optima. The basin of attraction is very wide so to find that global optima, RHC would have to run with many more restarts so that eventually it would stumble into the global optima. This however is basically a brute force approach. SA would find it difficult to ever find the global optima because in this implementation it only ever will move to its neighbors. This means that SA would have to have its initial state in the attraction basin of the global optima. Alternatively, if the temperature decay were to diminish sufficiently at the same time as it moves into the attraction basin of the global optima, SA could find the global optima. The odds of this occurring are very low and dealing with global optima with small basins of attraction is a clear issue for SA. Looking at the various decay schedules for SA, we see that changing the decay schedule does not ever allow SA to find the global optima, it just slows the overall time to reach the convergence. Additionally, the slower temperature decay schedule adds a lot of variance between runs.

In Figure 9, we can see that adjusting the number of restarts largely has no effect on the expected convergence of RHC. Although there was one run that was particularly lucky in restarts=4, all values of restarts converge on the local optima instead of the global optima because the basin of attraction is

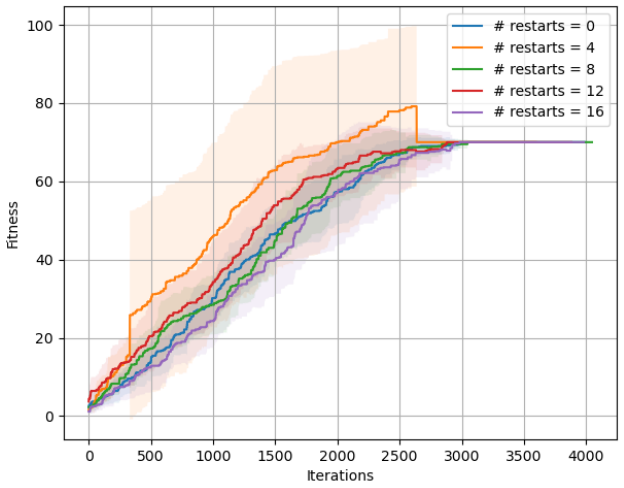


Fig. 9: Plot of Random Hill Climbing with various numbers of restarts on FourPeaks.

too small for the global optima. RHC will almost always get stuck in the local optima because the basin of attraction is so much bigger.

Unlike SA, which has a decaying exploration parameter, GA has persistent exploration using mutation. The mutation rate does not decrease and so GA continues to explore the problem space until it converges. In Table IV, we can see that as the mutation rate increase, the number of function evaluations decreases showing that the problem space is explored more quickly. Additionally, the average fitness increases showing that the exploration led to finding the global optima more consistently.

| Pop Size | Total Fxn Evals | Run Time (sec) | Time / Iterations (ms) | Fxn Evals / Iteration | Avg Fitness | Fitness Std |
|---|---|---|---|---|---|---|
| Hyperparameter | Total Fxn Evals | Convergence Time (sec) | Time / Iterations (ms) | Fxn Evals / Iteration | Fitness Avg | Fitness Variance |
| 288975 | 28.59 | 9.99 | 101.0 | 77.86 | 20.79 | 0.02 |
| 0.05 | 155538 | 16.91 | 10.98 | 101.0 | 85.71 | 26.84 |
| 0.1 | 137476 | 15.29 | 11.23 | 100.9 | 85.71 | 26.84 |
| 0.2 | 119510 | 13.23 | 11.17 | 100.9 | 85.71 | 26.84 |

TABLE IV: Table of Genetic Algorithm with various mutation rates on FourPeaks.

Population size can have a similar exploration effect, but in Table V we see that although the average fitness does increase, the number of function evaluations increases dramatically. In this problem space, increasing the mutation rate is a better exploration method.

In Table IX, we can see that increasing the population size increases the average fitness, but it also dramatically increases the number of function evaluations and the overall runtime. A higher population size allows MIMIC to more accurately model the problem space because there are more samples and

| Pop Size | Total Fxn Evals | Run Time (sec) | Time / Iterations (ms) | Fxn Evals / Iteration | Avg Fitness | Fitness Std |
|---|---|---|---|---|---|---|
| 50 | 90213 | 9.61 | 5.43 | 51.0 | 85.71 | 26.84 |
| 200 | 238814 | 24.24 | 20.39 | 200.9 | 85.71 | 26.84 |
| 300 | 331037 | 32.01 | 29.08 | 300.7 | 101.43 | 29.40 |
| 400 | 449490 | 43.57 | 38.84 | 400.7 | 101.43 | 29.40 |

TABLE V: Table of Genetic Algorithm with various population sizes on FourPeaks.

so more opportunity for some of the samples to land in the attraction basin of the global optima.

| Pop Size | Total Fxn Evals | Run Time (sec) | Time / Iterations (ms) | Fxn Evals / Iteration | Avg Fitness | Fitness Std |
|---|---|---|---|---|---|---|
| 50 | 5566 | 3.56 | 32.29 | 50.5 | 8.57 | 2.64 |
| 200 | 21164 | 7.05 | 66.32 | 199.1 | 16.57 | 3.64 |
| 300 | 32598 | 9.17 | 83.93 | 298.3 | 52.86 | 35.73 |
| 400 | 44116 | 11.35 | 102.25 | 397.4 | 55.29 | 31.83 |

TABLE VI: Table of MIMIC with various population sizes on FourPeaks.

| Algorithm | Total Fxn Evals | Convergence Time (sec) | Time / Iterations (ms) | Fxn Evals / Iterations |
|---|---|---|---|---|
| RHC | 3316 | 2.06 | 0.63 | 1.0 |
| SA | 4140 | 0.25 | 0.08 | 0.7 |
| GA | 180896 | 28.53 | 47.39 | 300.5 |
| MIMIC | 154208 | 64.10 | 124.88 | 300.4 |

TABLE VII: Table of Function Evaluations Statistics for Various Algorithms on Four Peaks .

### *E. KColors - MIMIC*

KColors is an optimization problem where the fitness is defined as the number of connected nodes that share the same color. In this discrete space, there is only two colors, and the nodes can have many connections. This fitness function is highly dependent on the structure of features. I expect that MIMIC will be able to identify this structure quickly because it is estimating the underlying probability distribution.

From Figure 10, we can see that GA both achieve high fitness values rather quickly, while SA and RHC both struggle to find the global optima.

Both SA and RHC struggle to reach the global optima. This is largely due to the fact that this problem space has many local optima which makes it difficult for them to escape their attraction basins. For RHC and SA, the convergence criteria calculations remain the same as the other problem spaces and are determined using Equation 1. Looking at Figure 10, we can see that SA and RHC reach their optima and have no further increase in fitness. This shows that the max_attempts, or the convergence criteria was set correctly. GA and MIMIC were again selected to be a fraction of the max_attempts calculated for RHC and SA.

Where MIMIC shines is understanding the underlying structure of a problem space. Because MIMIC has this understanding, it can achieve convergence with fewer function

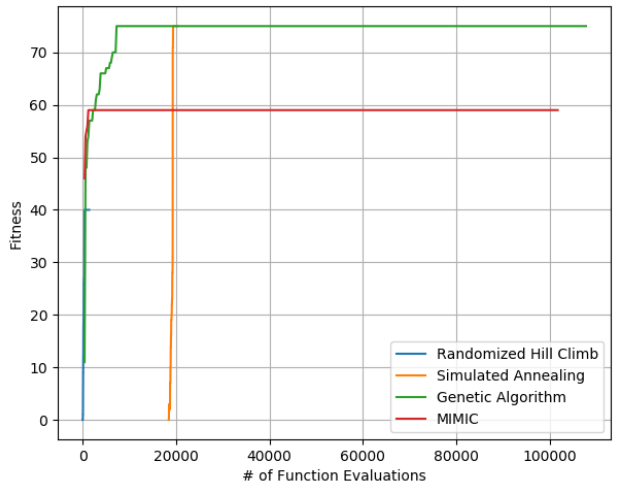


Fig. 10: Plot of Fitness vs Iterations of Various Algorithms, on KColors.

evaluations. It isn't able to reach the same maximum that GA is able to. In the short term however, it can. While that might not be so important in this context, some function evaluations could be very complex in time and space. In Table XI, you can see that MIMIC is requiring less function evaluations than GA. In a very similar manner to FourPeaks, increasing population size increases the wall clock time and function evaluations, but does result in a more consistent mean fitness score.

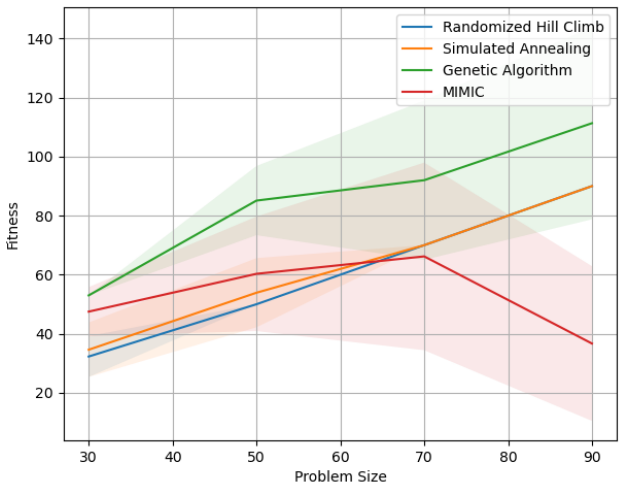


Fig. 11: Plot of Fitness vs Complexity of Various Algorithms, on KColors.

| Pop Size | Total Fxn Evals | Run Time (sec) | Time / Iterations (ms) | Fxn Evals / Iteration | Avg Fitness | Fitness Std |
|---|---|---|---|---|---|---|
| 100 | 101406 | 15.07 | 14.99 | 100.9 | 19.00 | 0.00 |
| 300 | 304017 | 26.88 | 26.59 | 300.7 | 28.00 | 0.00 |
| 500 | 571151 | 43.14 | 37.81 | 500.6 | 44.00 | 0.00 |

TABLE VIII: Table of MIMIC with various population sizes on KColors.

Similar to MIMIC, increasing population size for GA increases the function evaluations and the overall wall clock time, but the increase in fitness score is relatively small and there is less value in increasing the population for GA in this problem.

| Pop Size | Total Fxn Evals | Run Time (sec) | Time / Iterations (ms) | Fxn Evals / Iteration | Avg Fitness | Fitness Std |
|---|---|---|---|---|---|---|
| 50 | 53314 | 3.42 | 3.27 | 51.0 | 35.00 | 0.00 |
| 100 | 104654 | 6.71 | 6.47 | 100.9 | 47.00 | 0.00 |
| 200 | 208856 | 13.57 | 13.05 | 200.8 | 47.00 | 0.00 |

TABLE IX: Table of MIMIC with various population sizes on KColors.

Using different restarts on RHC is fruitless as there is no increase in score (Table X). The complex relationships in kcolors are impossible to optimize for RHC.

| Restarts | Total Fxn Evals | Run Time (sec) | Time / Iterations (ms) | Fxn Evals / Iteration | Avg Fitness | Fitness Std |
|---|---|---|---|---|---|---|
| 0 | 1437 | 0.05 | 0.03 | 1.0 | 32.00 | 0.00 |
| 8 | 1719 | 0.47 | 0.27 | 1.0 | 47.00 | 0.00 |
| 16 | 1719 | 0.86 | 0.50 | 1.0 | 47.00 | 0.00 |

TABLE X: Table of Random Hill Climbing with various restart values on KColors.

In a similar reasoning to RHC, SA is also not very useful on kcolors because of the complexity of the relationships. For this reason, altering the decay schedule does little to affect the overall score as seen in Figure 12.

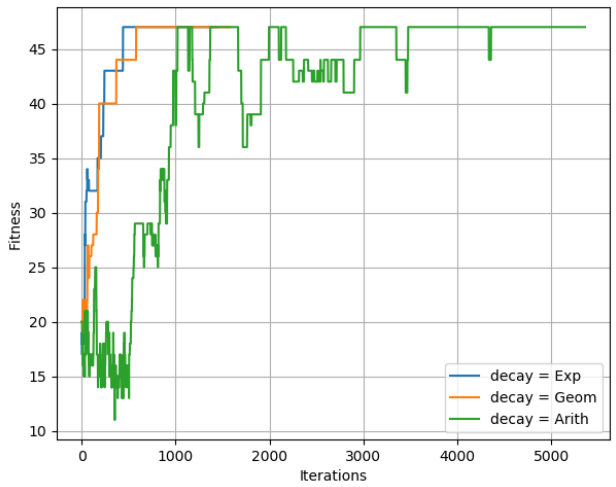


Fig. 12: Plot of Fitness vs Iterations of decay methods for Simulated Annealing, on kcolors.

## II. Neural Network Weights Optimization

In this section we will attempt to use Random Optimization algorithms for updating the weights of a neural network. This is a difficult problem because the nodes are all connected, the changes in weights of one node, will have an affect on all the nodes after it. This is an incredibly complex function with

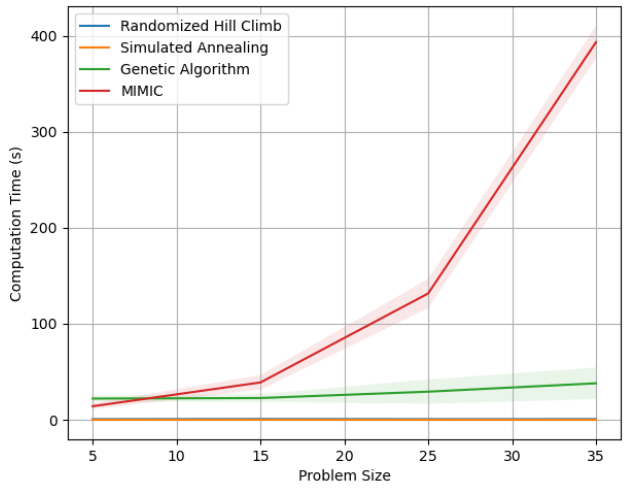


Fig. 13: Plot of Computation Time vs Complexity of Various Algorithms, on KColors.

| Algorithm | Total Fxn Evals | Convergence Time (sec) | Time / Iterations (ms) | Fxn Evals / Iterations |
|---|---|---|---|---|
| RHC | 3316 | 2.06 | 0.63 | 1.0 |
| SA | 4140 | 0.25 | 0.08 | 0.7 |
| GA | 180896 | 28.53 | 47.39 | 0.0 |
| MIMIC | 154208 | 64.10 | 124.88 | 0.0 |

TABLE XI: Table of Function Evaluations Statistics for Various Algorithms on KColors.

many interconnnected relationships with many local optima. Additionally, the weights represent a continuous optimization space which is more difficult to solve with random optimization. Given the many local optima, I expect that SA and RHC will not be able to reduce loss and will result in poor classification scores of both the training set and testing sets and there for have low bias and high variance. I expect that GA should be able to move past some of the local optima and have higher bias and lower variance than SA and RHC. However, Gradient Descent should out perform all of the RO methods. This is because it is purpose built for this application and has inductive bias to increase bias and reduce variance.

### A. Methods

All of the following experiments will use a fixed architecture of two layers of 50 nodes each. The data set is the Red Wine dataset from the previous assignment.

### B. Data Pre-Processing

The data is scaled based on the training data and the applied to the test data so that the neural networks can use the data properly. Because we are only scaling based on the training data, we are not biasing to the test data.

We use a weighted f1 score to fairly score partially unbalanced fold. We used stratified shuffle folds to ensure that there is an equal number of labels in each fold. We split the data set into a test and train set. The training set will be used for hyper-parameter validation.

### C. Results

In Figure 14 we can see that Gradient Descent (GD) has high bias at lower values of training samples, but this is reduced as training samples increase. This is expected behavior because with fewer samples, there are less examples per class and GD is able to reduce the loss to fit the few examples perfectly. As more examples of the classes are added to the training set, GD is forced to generalize its classification and thus reduce its bias which can be seen by the increasing score. Even with the full training set, GD still has some bias and some variance. This variance can be seen by observing the difference between the training score and the cross-validation score. In hindsight, this could have been remedied by simplifying the architecture to reduce the overfitting to the training data. Having a simpler architecture would lead to better generalization. The wall clock time for GD scales linearly with the number of samples as the overhead of classifying the training samples with the neural network drives the overall fit time.

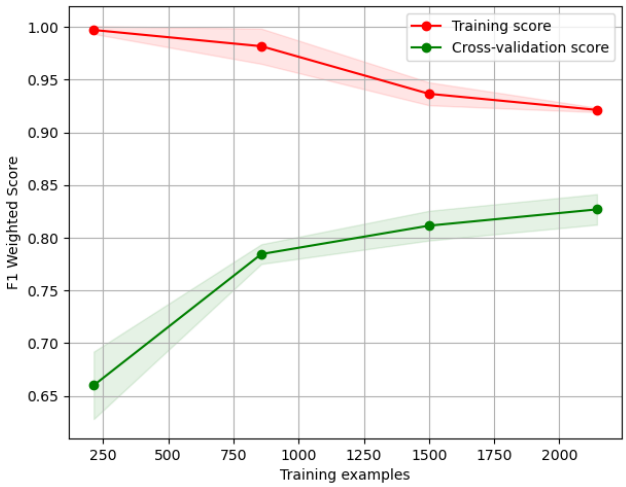


Fig. 14: Learning Curve of Neural Network using Gradient Descent on Red Wine Database.

In Figure 15, we see that initially there is a lot of variation in the training score and the cross validation score. This means the model is over fitting to the training data. This could mean that there were easy features to optimize in the training set, but led to lower performance in the validation set. This indicates high variance. The low score indicates that GA produces a model with a high bias, but with more training examples it is able to reduce its bias and increase its score. The reduction in the deficit between the training score and the validation score at full training sample sizes indicates that GA has reduced the variance in the model. The lower standard deviations in test scores also highlight that there has been a reduction in variance. The wall clock time for GA is quite poor compared to the other algorithms. This is because GA is essentially training a different model for each sample in the population size. While this means that GA is able to avoid more local optima, it comes at a large order

of magnitude time cost. Looking at Table XII, we can see that the learning rate has little affect on the overall scores of the GA optimization function. I thought that the learning rate would allow the genetic algorithm to make larger changes and search the problem space properly. Likely the problem space is too complex and that a greater understanding of the relationships between the different features is required. GA is unable to capture complex relationships between different features unlike GD.

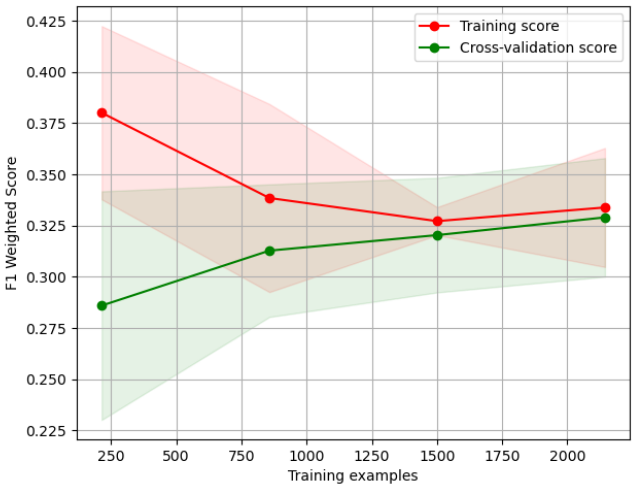


Fig. 15: Learning Curve of Neural Network using Genetic Algorithm on Red Wine Database.

| learning rate | mean test score | std test score | mean train score | std train score | fit time |
|---|---|---|---|---|---|
| 0.0001 | 0.522 | 0.038 | 0.531 | 0.006 | 913 |
| 0.01 | 0.521 | 0.037 | 0.530 | 0.006 | 907 |
| 0.1 | 0.523 | 0.038 | 0.531 | 0.006 | 873 |

TABLE XII: Table of Scores and Fit times for various learning rates for Genetic Algorithms on Red Wine dataset. Results of 5 fold cross validation.

Simulated annealing gets quite terrible scores with only slight improvement (Figure 16). This shows that SA produces models with very high bias with little improvement with an increase in training samples. There is little to no difference in the mean scores between the cross validation set and the training set, and there little amounts a variation in these scores. It has a similar amount of score variation to gradient descent, although it looks larger due to the scaling. This means that using SA the model is not sensitive to fluctuations in training data and thus has a low variance. This makes sense because SA is likely being trapped in a large basin of attraction that is related to features seen in much of the training data. Looking at the Figure 18, we can see that SA is not able to reduce its loss even after many iterations. This supports the idea that SA is stuck in a basin of attraction.

To avoid getting stuck in these basin, adjusting hyperparameters to encourage exploration could help discover higher optima. In Table XIII we see that despite adjusting the initial temperature by large orders of magnitude as well as altering the learning rate, the scores remain essentially the same. This makes sense because the problem space is very non-linear with many spikes and basins of attraction that make it particularly difficult for SA. Similar to Four Peaks, its difficult for SA to overcome these local optima and it is essentially random chance if the algorithm is able to move to a smaller basin of attraction that has a larger local optima. This problem is further compounded by the way that SA is implemented in this continuous problem space. When SA looks for a random neighbor, it does so in increments equal to the size of the learning rate. Increasing the learning rate increases SA's exploration, but reduces its ability to optimize. If the width of the optima is smaller than the learning rate, SA could step over the optima entirely. One small bright spot with SA is that because it only runs one function evaluation per iteration, its overall wall clock time remains very low and is actually faster than GD which makes sense because SA doesn't have do run any back propagation calculations, just the fitness evaluation.

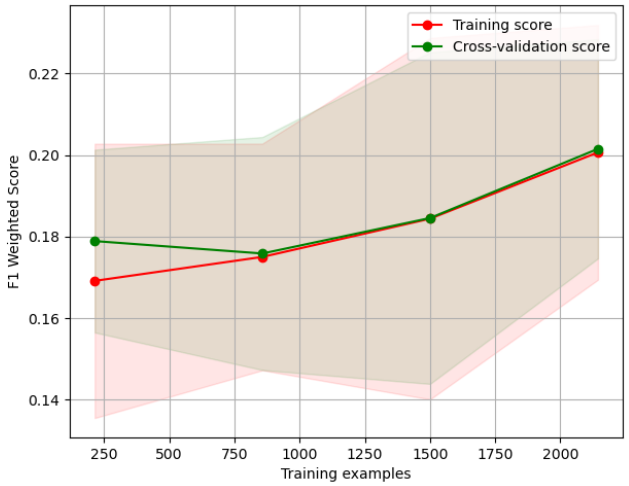


Fig. 16: Learning Curve of Neural Network using Simulated Annealing on Red Wine Database.

| learning rate | initial temp | mean test score | std test score | mean train score | std train score | fit time |
|---|---|---|---|---|---|---|
| 0.0001 | 1 | 0.521 | 0.0300 | 0.532 | 0.0054 | 175 |
| 0.0001 | 100 | 0.504 | 0.0224 | 0.522 | 0.0091 | 174 |
| 0.0001 | 10000 | 0.510 | 0.0211 | 0.511 | 0.0063 | 176 |
| 0.001 | 1 | 0.521 | 0.0300 | 0.532 | 0.0054 | 174 |
| 0.001 | 100 | 0.504 | 0.0224 | 0.522 | 0.0091 | 175 |
| 0.001 | 10000 | 0.510 | 0.0211 | 0.511 | 0.0063 | 176 |
| 0.1 | 1 | 0.521 | 0.0300 | 0.532 | 0.0054 | 174 |
| 0.1 | 100 | 0.504 | 0.0224 | 0.522 | 0.0091 | 174 |
| 0.1 | 10000 | 0.510 | 0.0211 | 0.511 | 0.0063 | 175 |

TABLE XIII: Table of Scores and Fit times for various hyperparameters for Simulated Annealing on Red Wine dataset. Results of 5 fold cross validation.

In RHC, we shouldn't expect to see results that are better than SA as they both suffer from the same structure problem exposed in Four Peaks with local optima with large basins of attraction. Indeed, looking at Figure 17, we see that like SA, RHC has terrible scores and a large bias. RHC is able to improve is slightly with more training samples, but it

still has a lower overall score and thus higher bias than SA. RHC does have low variation in training and cross validation scores which suggests that the model that RHC produces has low variance. The difference between the training and cross validation score is low which also indicates a low variance model. This is again similar to SA where RHC is likely to get stuck in a large basins with a local minima. This claim is supported by the loss curve in Figure 18 where RHC reduces loss in the first few iterations, but then fails to further reduce the loss. Another consideration is that in this implementation, RHC takes steps of size equal to the learning rate. This means that if there is a optima with basins that are smaller than the learning rate, RHC has no chance of finding them because it will step right over the optima. Reducing the learning rate will give it a chance of finding it, but it will also be more likely to get stuck in a local optima. Similar to SA, RHC has a decent wall clock time as it is only running one fitness evaluation per iteration with no additional calculations.

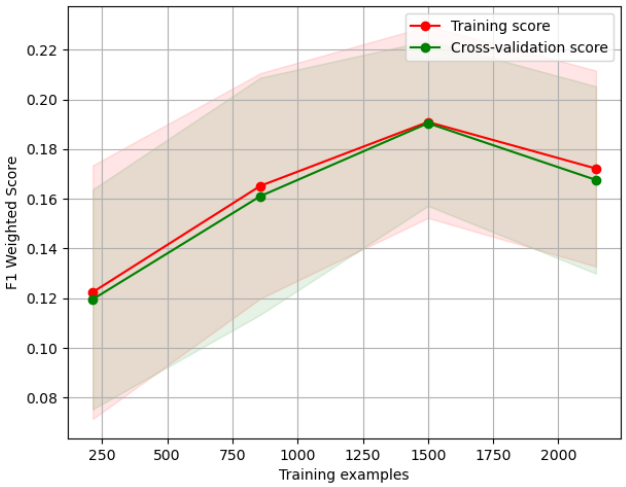


Fig. 17: Learning Curve of Neural Network using Random Hill Climbing on Red Wine Database.

For RHC a grid search was performed with 'learning_rate': [0.01, 0.001] and 'restarts': [5, 25, 50] to determine optimal hyper parameters. We can see that increasing the learning rate has a negligible effect on the bias and variance of this neural network. Showing that there are many local optima with large basins of attraction that catch RHC despite varying step size. By increasing restarts, the score does increase and reduces the bias of the model, but this comes at a large time cost that scales dramatically.

Gradient descent (GD) performs much better because this algorithm was purpose built with complete knowledge of the underlying structure behind the problem it aims to solve. Gradient descent performs a derivative of the loss function with respect to the input for each node at each layer. This gives gradient descent a huge advantage because it is directly able to attribute the relationship between a given feature and the fitness outcome.

Each function evaluation in this problem is very computationally expensive. This means running the entire training set through the classifier. GD has another large advantage here because it only needs one function evaluation at each iteration while genetic algorithms needs a function evaluation for each of the samples in its population.

| learning rate | restarts | mean test score | std test score | mean train score | std train score | mean fit time | std fit time |
|---|---|---|---|---|---|---|---|
| 0.01 | 5 | 0.094 | 0.0097 | 0.093 | 0.0028 | 13.9 | 0.61 |
| 0.001 | 5 | 0.094 | 0.0097 | 0.093 | 0.0028 | 15.4 | 0.65 |
| 0.01 | 25 | 0.087 | 0.0127 | 0.086 | 0.0048 | 170.3 | 2.68 |
| 0.001 | 25 | 0.087 | 0.0127 | 0.086 | 0.0048 | 172.8 | 6.55 |
| 0.01 | 50 | 0.107 | 0.0136 | 0.107 | 0.0032 | 604.4 | 10.48 |
| 0.001 | 50 | 0.107 | 0.0136 | 0.107 | 0.0032 | 593.5 | 1.67 |

TABLE XIV: Table of Scores and Fit times for various hyperparameters for Hill Climb with Restarts on Red Wine dataset. Results of 5 fold cross validation.

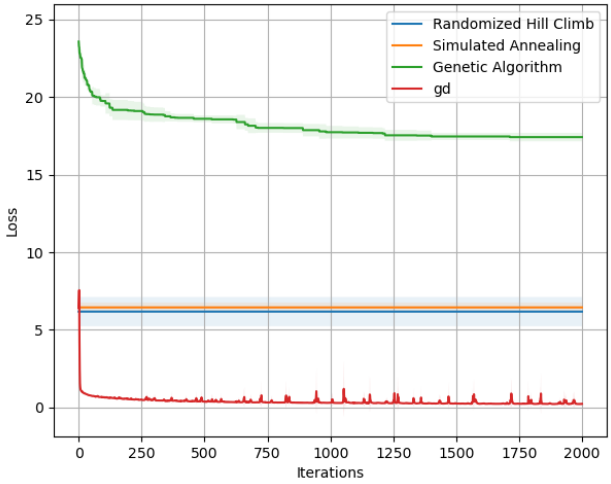


Fig. 18: Loss Curve of Neural Network Weight Optimization using various algorithms on Red Wine Database.

## III. Appendix

## References

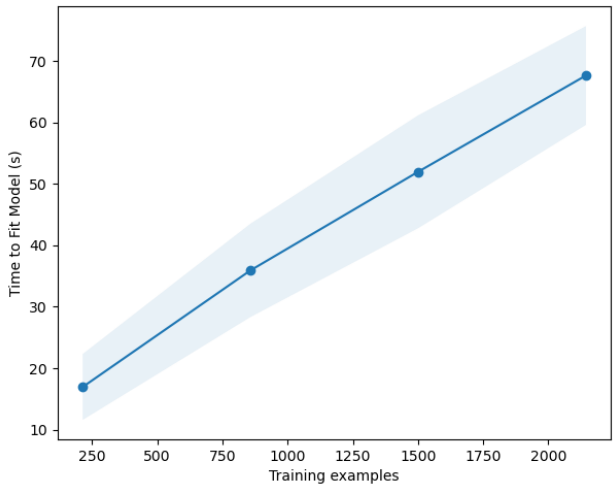


Fig. 19: Scalability of Neural Network using Gradient Descent on Red Wine Database.

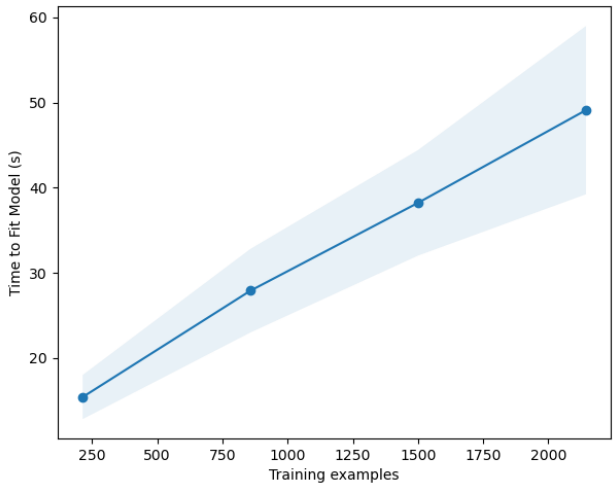


Fig. 20: Scalability of Neural Network using Random Hill Climbing on Red Wine Database.

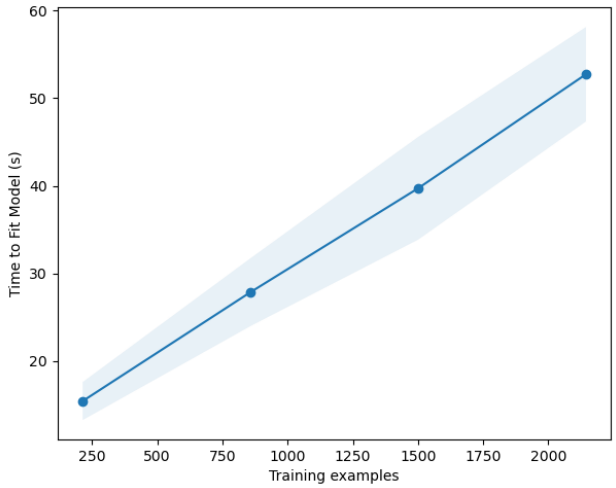


Fig. 21: Scalability of Neural Network using Simulated Annealing on Red Wine Database.

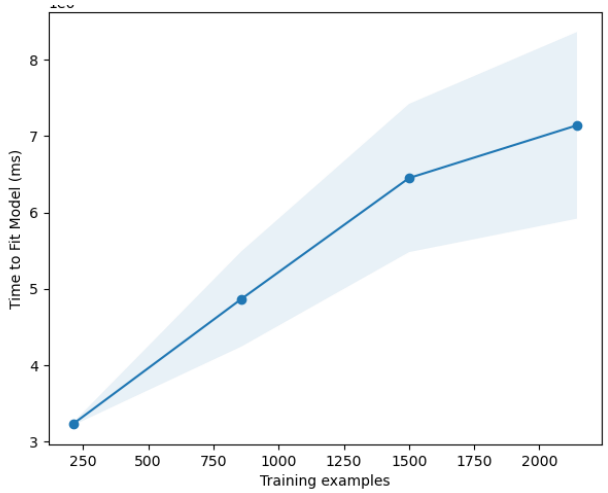


Fig. 22: Scalability of Neural Network using Genetic Algorithms on Red Wine Database.