CS7641 Assignment 2 – Randomized Optimization

Michael Lukacsko
mlukacsko3@gatech.edu

***Abstract*—This assignment has two parts. Part 1 implements 3 random optimization problems (flip-flop, continuous peaks, and four peaks) using 4 local random search algorithms (randomized hill climbing, simulated annealing, genetic algorithm, and MIMIC). Each problem will be highlighted by the algorithm that performs the best and why. Part 2 will use the first three local random search algorithms (randomized hill climbing, simulated annealing, and a genetic algorithm) to find optimal weights for a neural network and compared to a neural network that using backpropagation for optimization.**

***Keywords*— randomized hill climbing, simulated annealing, genetic algorithm, MIMIC, Neural Network, flip-flop, continuous peaks, four peaks.**

I. Optimization problem introduction

A. Flip-flop optimization problem

The flip-flop optimization problem is a type of combinatorial optimization problem that involves finding the optimal binary string by flipping consecutive bits in the string. The goal is to find the binary string that maximizes or minimizes a given objective function.

Formally, given a binary string of length n, the flip-flop optimization problem involves selecting a pair of adjacent bits in the string and swapping them, with the aim of finding the binary string that either maximizes or minimizes a given objective function f(x), where x is the binary string.

B. Continuous Peaks optimization problem

The continuous peaks optimization problem is another type of combinatorial optimization problem that involves finding the longest sequence of consecutive values that are greater than a threshold value in a binary string. This problem is challenging because the fitness function has many local maxima, making it difficult for optimization algorithms to find the global maximum. Moreover, the problem exhibits both discrete and continuous characteristics, making it difficult to use standard optimization techniques.

C. Four Peaks optimization problem

The Four Peaks optimization problem is like the continuous peaks problem in the sense it is another combinatorial optimization problem. The problem involves finding the maximum number of consecutive values that are greater than a threshold value in a binary string. However, unlike the continuous peaks problem, the four peaks problem has two distinct peaks, each consisting of a sequence of consecutive 1's in the binary string. The two peaks can be separated by a sequence of consecutive 0's or a random sequence of 1's and 0's.

II. Search algorithm introduction

A. Randomized Hill climbing (RHC)

RHC is an optimization algorithm that can be used to solve a wide range of optimization problems. The algorithm starts by randomly generating an initial solution, and then repeatedly evaluates its fitness by comparing it with its neighboring solutions. If a better solution is found, it replaces the current solution with it and the process is repeated until no further improvements can be made.

B. Simulated Annealing (SA)

SA is a stochastic optimization algorithm used to find the global optimum of a problem. It is inspired by the physical process of annealing in metallurgy, where a metal is heated and then slowly cooled to increase its resistance to stress and improve its properties. SA starts with an initial solution and iteratively explores neighboring solutions by randomly perturbing the current solution. The algorithm uses a temperature parameter, which controls the degree of randomness in the search. At high temperatures, the algorithm is more likely to accept a worse solution, allowing it to explore a wider search space. As the temperature decreases, the algorithm becomes more greedy and focuses on improving the current solution.

Unlike Random Hill Climbing, SA accepts a worse solution using a probability that depends on the difference between the fitness of the current and proposed solutions and the current temperature. This allows the algorithm to escape from local optima and explore different regions of the search space.

C. Genetic Algorithm (GA)

GA is also a stochastic optimization algorithm; however, it is inspired by the process of natural selection in biology. It is used to find the global optimum of a problem by mimicking the process of evolution and survival of the fittest. It starts by creating an initial population of candidate solutions, which are represented as chromosomes. Each chromosome is made up of genes, which encode the values of the variables in the problem. The population is then iteratively evolved through a process of selection, crossover, and mutation.

D. MIMIC

MIMIC is a stochastic optimization algorithm belonging to the family of probabilistic graphical models, which represent the relationships between variables in a problem as a graph. MIMIC starts by creating an initial population of candidate solutions, which are represented as probability distributions over the variables in the problem. The algorithm then iteratively refines the population through a process of selection, estimation, and maximization.

III. Flip-flop analysis

A. Overview

Each of the 4 search algorithms are assessed individually and together. On their own, each algorithm is tuned using hyperparameters, and, as a group, the performance is examined with respect to fitness, computation time and the effect of increasing problem size.

1) *Randomized Hill Climbing (RHC)*

The RHC algorithm implemented here is tested with varying numbers of restarts. In essence, this is a hyperparameter tuning technique that triggers random restarts which run the algorithm multiple times and with different random initial starting points. This is done to help overcome a problem where the algorithm gets stuck in local optima's. Figure 1 below shows this RHC with restart values of 0,5, 10, and 20.

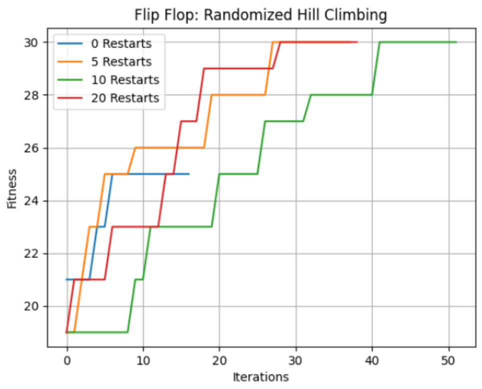

Figure 1 – Randomized Hill Climbing – Fitness vs Iterations

Based on the visual above, implementing this RHC with a restart value of 10 increases the computational cost the most because it requires the most iterations to find a suitable solution. Moreover, RHC implemented with 5,10 and 20 restarts find the optimal solution and have a fitness score of 30. Interestingly, as might be assumed, more restarts are not always better. In the plot above, 5 restarts and 20 restarts perform the best and are able to solve this optimization problem around the 27th iteration. This is because, in some cases, performing fewer restarts can lead to better performance because it forces the algorithm to explore a smaller region. Lastly, using 0 restarts results in termination of the RHC algorithm with a suboptimal fitness score of about 25. It can be assumed that the initial solution of this RHC with 0 restarts was either poor or there were many local optima. As a result, the algorithm got stuck in a suboptimal solution and failed to find the global optimum.

2) *Simulated Annealing (SA)*

This SA algorithm is tested with three different cooling schedules: geometric, arithmetic, and exponential decay. Because SA accepts worse solutions with a certain probability, which allows it to jump out of local optima and explore new regions, the fitness score fluctuates, and creates a plot that looks very different than what is plotted for the other 3 algorithms. The results of this are plotted below in figure 2.

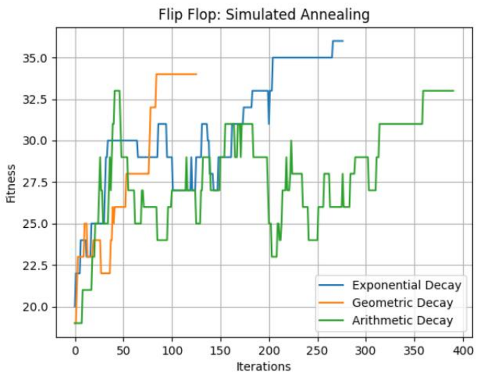

Figure 2 – Simulated Annealing – Fitness vs Iterations

The image above shows that SA implemented with geometric decay terminates first, followed by exponential decay, and arithmetic decay. This is expected because, in terms of aggressiveness, the most aggressive decay schedule is the geometric decay schedule, which reduces the temperature at each iteration by multiplying it by a constant factor. In general, this decay schedule reduces the temperature exponentially over time and can be effective in quickly reducing the search space and finding a good solution. However, in this case, it causes the SA algorithm to converge too quickly to a suboptimal solution. The arithmetic decay schedule takes the most iterations to terminate, and, in doing so, has the lowest fitness score. Exponential decay terminates in between the other two and has the greatest fitness score. This makes it the better choice in solving this optimization problem.

3) *Genetic Algorithm (GA)*

This GA algorithm is fitted to the flip-flop optimization problem with varying mutation rates (0.1, 0.3, and 0.5) and population sizes (50, 250, and 500). The figure below is plotted to evaluate its performance using the fitness score vs number iterations.

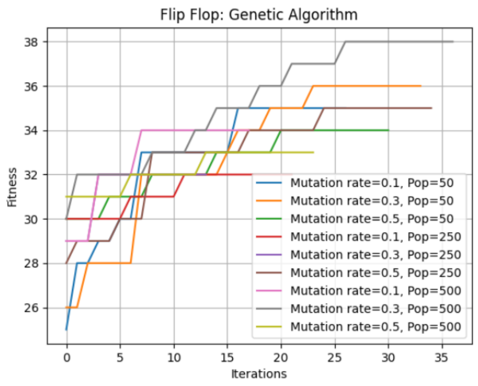

Figure 3 – Genetic Algorithm – Fitness vs Iterations

The plot above shows this algorithm taking the greatest number of iterations overall to reach an optimal solution. Where the population = 50, and mutation rate = 0.3 (30%), the fitness score of 36 comes close to the best fitness score, 38, which is reached when the population = 500 and mutation rate is also 30%. Considering the fact that as the population increases the computational cost does as well, it's clear that, for this optimization problem, a population =50 and mutation rate = 30% is optimal.

4) *MIMIC*

The MIMIC algorithm is implemented with varying population sizes and percentage of samples used to update the probability model. In theory, larger populations should help explore the solution space more thoroughly and avoid getting stuck in local optima. Conversely, it also increases the computational cost. As a result, higher percentage values should help the algorithm converge faster and generate more diverse solutions, but also increase the computational cost of the algorithm. The figure below is plotted using "keep" percentages equal to 0.1, 0.3, and 0.5. The "pop" (population) size has a range of 50, 250, and 500.

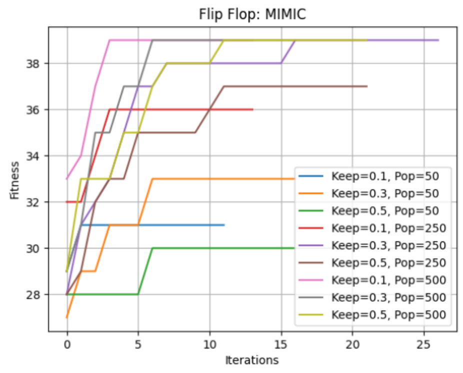

Figure 4 - MIMIC – Fitness vs Iterations

The image above makes clear that, in general, the fitness score is greatest when the population size is the largest at 500. Interestingly, this is not the case where the population size = 250 and the keep percentage = 0.3 (30%). Here, the optimal solution is reached, however, it takes more than 15 iterations. Where the population = 500, a keep percentage of 0.1 (10%) finds the optimal solution in less than 5 iterations making it the most effective hyperparameter for this optimization problem.

5) *Algorithm Comparison*

To determine the best algorithm for this optimization problem, fitness with respect to iterations and problem size is explored, as well as the amount of time to compute an optimal solution. Figure 5 below is a visual representation of performance. On the left, all 4 algorithms performance is plotted with a varying problem size. On the right, iterations are the performance metric with respect to fitness.

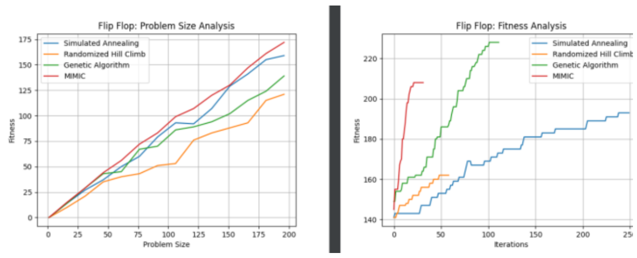

Figure 5 – Performance Analysis – Problem Size, left. Iterations, right

Given the plots above, it would be safe to say that MIMIC is the best suited local random search algorithm for the flip flop optimization problem. This is because it outperforms the other three random search algorithms for all problem sizes. This would be true; however, figure 6 below shows that the MIMIC algorithm also takes about 7 times longer to compute the optimal solution when compared to the next slowest algorithm, GA. Additionally, the GA algorithm does not perform significantly worse than MIMIC does when compared in the performance plots above. As a result, looking at the fitness score on the right side of the image above, GA outperforms the other 3 algorithms, does well with varying problems sizes, and is able to compute the optimal solution in under 1 second. Hence, for this optimization problem, GA is the highlighted algorithm.

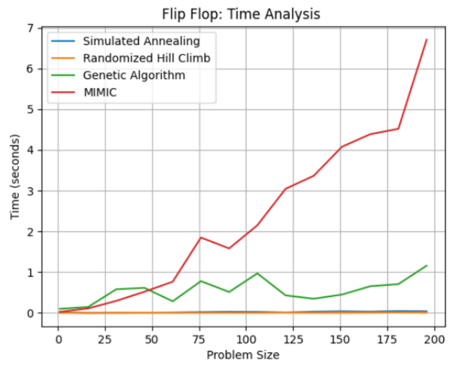

Figure 6 - Time Analysis

IV. CONTINUOUS PEAKS ANALYSIS

A. *Overview*

Again, each of the 4 search algorithms are assessed individually and as a group. For this optimization problem, the problem size (bit string) is on the smaller side because of the amount of time needed to compute a solution. This is explored in more detail below. In addition to time, performance is assessed with respect to fitness and problem size.

1) *Randomized Hill Climbing (RHC)*

This RHC algorithm is not well suited for the continuous peaks problem. Because RHC only accepts perturbations that improve the objective function value locally, without considering the global picture, it is prone to getting stuck in local maxima. As a result, RHC is more effective on low-dimensional discrete search spaces where each perturbation can be fully explored and evaluated.

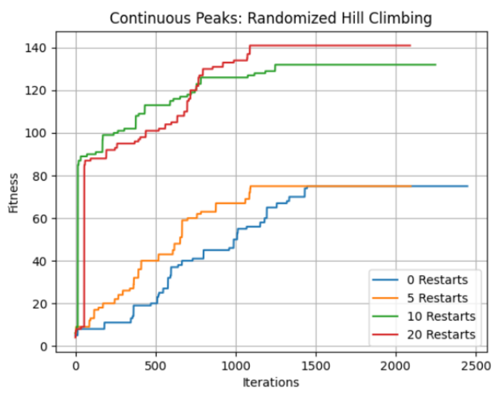

Figure 7 - Randomized Hill Climbing – Fitness vs Iterations

The image above shows the ineffectiveness of RHC on this problem and is especially apparent where the number of restarts < 10 and increases of the fitness score are much less dramatic. Where the number of restarts is >= 10, it takes more than 1000 iterations before the maximum fitness score is reached. This is in addition to the fact that the fitness score rapidly increases in the first 100 iterations. This highlights the limitations of this algorithm on this particular optimization problem.

2) *Simulated Annealing (SA)*

Implemented with three different decay schedules, this algorithm solves the continuous peaks optimization problem effectively. Also, unlike the flip flop problem, this SA implementation is able to avoid being hung in a local optimum. However, compared to the flip flop problem, the number of iterations needed to find a solution is greater. Figure 8 below shows this.

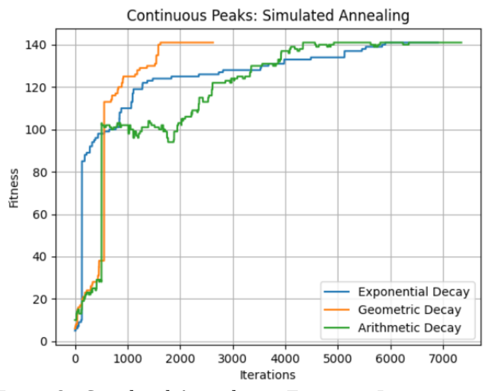

Figure 8 - Simulated Annealing – Fitness vs Iterations

Referencing the plot above, it's clear that a geometric decay schedule is best suited for this problem. This results from the fact that a geometric decay schedule allows the algorithm to explore the search space effectively by starting with a high temperature and gradually decreasing it. This enables the algorithm to find the global optimum by effectively exploring the entire search space, which is critical for this problem. Regarding the exponential and arithmetic decay schedules, both are slower to converge because the temperature decreases more slowly at the beginning of the search for these decay schedules. This in turn leads to more exploration of the search space and slower convergence. Figure 8 above makes this clear as exponential and arithmetic decay schedules require around 4000 more iterations to reach the same fitness score that geometric decay reached in around 1750 iterations.

3) *Genetic Algorithm (GA)*

This GA algorithm performs very similarly with varying mutation rates and population sizes. Except for a mutation rate = 0.1 and population size of 50, this GA can find an optimal solution to this problem in less than 1000 iterations. A mutation rate of 0.1 and population size of 50 is unable to find the optimal solution. This observation, and the plot below, show just how similarly this algorithm performs with different mutation rates and population sizes because the search space of the problem is relatively small, and the objective function is simple. As a result, a smaller population size proves to be sufficient at exploring the search space and finding the global optimum.

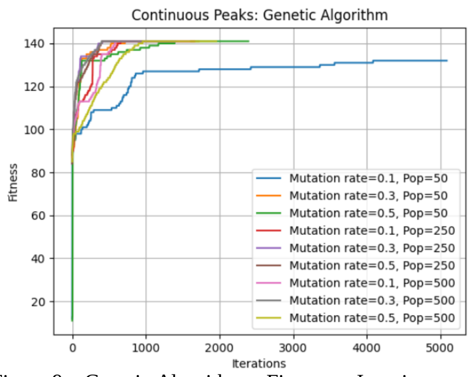

Figure 9 – Genetic Algorithm – Fitness vs Iterations

4) *MIMIC*

This MIMIC algorithm performs very well, as can been seen in the plot below. Furthermore, the plot below shows that the fitness score increases where higher population size and higher keep percentage hyperparameters are used.

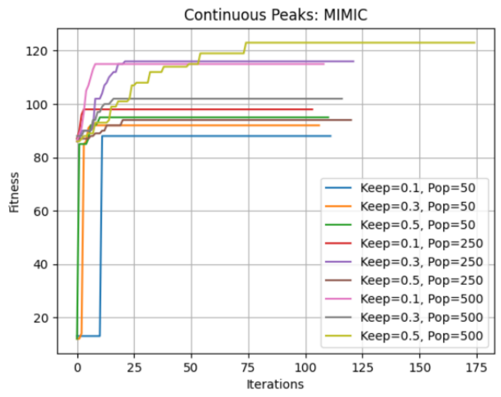

Figure 10 - MIMIC – Fitness vs Iterations

In the case of continuous peaks, it is important to strike a balance between exploration and exploitation to avoid getting stuck in local optima. A larger population size helps with exploration, while a higher keep percentage value helps with exploitation. Hence, the resulting plot.

5) *Algorithm Comparison*

Referencing figure 11 below, it is evident that both the Random Hill Climbing (RHC) and MIMIC algorithms do not perform as well as the Genetic algorithm (GA) and Simulated Annealing (SA) algorithm. In the problem size analysis pictured on the left, SA and GA perform the same for this optimization problem. However, GA finds the optimal solution in less iterations. This is plotted in the fitness analysis figure on the right below.

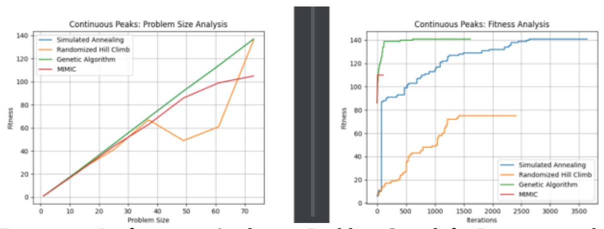

Figure 11 - Performance Analysis – Problem Size, left. Iterations, right

Building on the observations highlighted above, SA is the best choice for this problem. Figure 12 below shows that GA takes more than 20 seconds to find the optimal solution while SA does it in about 1 second for the same range. As it relates to the continuous peaks problem, the search space is large, and evaluating the fitness function for each candidate solution can be computationally expensive. As a result, GA requires more iterations to converge to a good solution, and each iteration takes longer due to the time needed to evaluate the fitness function.

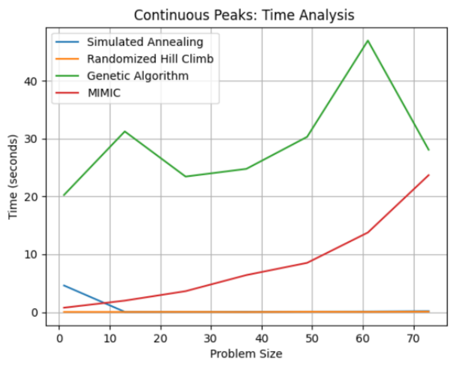

Figure 12 – Time Analysis

V. FOUR PEAKS ANALYSIS

A. *Overview*

The same 4 search algorithms are used on this problem to find a global maximum of a function with multiple local maxima and a deceptive landscape. Gauging the effectiveness of these algorithms will be determined by which one is best able to balance the tradeoff between exploration and exploitation. Like before, this will be done by evaluating each of the search algorithms fitness as it relates to problem size as well as the time to compute a solution.

1) *Randomized Hill Climbing (RHC)*

The RHC algorithm is a well-suited search algorithm for this problem, however, it can be limited by the optimization problems deceptive language. To increase RHC effectiveness, this implementation is augmented with a random restart mechanism, where the algorithm is run multiple times with different random starting points. Like the other tests, the number of restarts ranges between 0 and 20. Figure 13 below shows the result of this test.

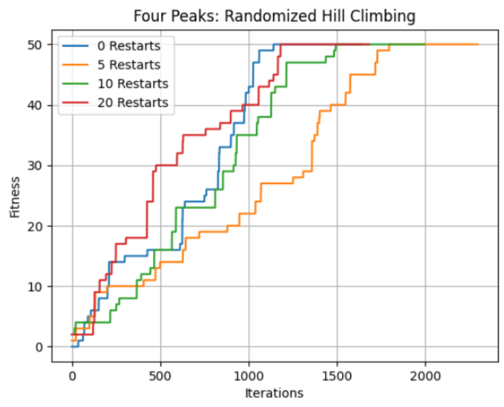

Figure 13 – Randomized Hill Climbing – Fitness vs Iterations

The result supports the theory that this is a well-suited algorithm for this problem. This is seen in the plot above where, regardless of the number of restarts, a solution is reached. Moreover, this RHC implementation avoided getting stuck in a local optimum. The plot also shows that performance is best where the restart parameter is set to 20. Supporting this is the fact that the fitness score increases the sharpest with respect to the number of iterations. The second best is when the number of restarts = 0, meaning that this RHC implementation was able to find the global maximum and avoid getting stuck in a local optimum without needing to restart from a different random point. Where the number of restarts = 5, this RHC algorithm performs the worst. In this case, it is possible the number of local maxima required more exploration to solve this problem.

2) *Simulated Annealing (SA)*

SA also performs well on this optimization problem finding the global maximum in at most 1500 iterations, depending on the decay schedule utilized. Figure 14 below highlights the performance of this algorithm using three different decay schedules.

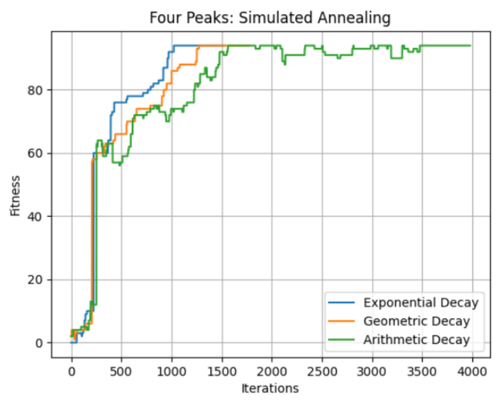

Figure 14 – Simulated Annealing – Fitness vs Iterations

Because the solution space for the Four Peaks problem is rugged, with many local maxima, it is important for the SA algorithm to explore the solution space effectively to find the global maximum. An exponential decay schedule tends to allow for more exploration early on in the algorithm by starting with a high temperature and gradually reducing the temperature. This allows the algorithm to escape local maxima and explore more of the solution space. The plot above shows the three decay schedules working similarly in finding the global maxima, however, the exponential decay schedule finds the solution at around 1000 iterations while the other two at about 1250 and 1500 iterations.

3) *Genetic Algorithm (GA)*

This GA solved the Four Peaks problem very efficiently. By varying the algorithms population of candidates and mutation rate, it can be seen in the plot below that GA was able to find a solution in the least number of iterations compared to the other two algorithms discussed thus far.

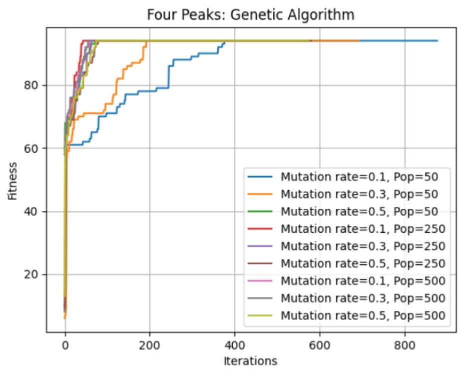

Figure 15 – Genetic Algorithm – Fitness vs Iterations

Most notable in figure 15 above is when the population is given a value greater than 50. When assigned a value of 250 or 500, the importance of the mutation rate is almost negated. This can be seen where the population is 250 and 500, and the mutation rate varies between 0.01 and 0.5. In all these cases, this GA algorithm solved the problem in less than 200 iterations, which is far superior that the other algorithms being used. As such, the larger population sizes help the algorithm explore a wider range of solutions and find a solution more quickly. However, where the population has a value of 50 and mutation rate of 0.01 or 0.3, this GA algorithm does not perform quite as well. Where these values are applied, the GA gets stuck in local optima and creates the need for more iterations to solve this optimization problem.

4) *MIMIC*

Where this MIMIC algorithm has the highest fitness scores, it was able to find a solution very quickly. This is a signal that this MIMIC algorithm is an effective algorithm at solving this optimization problem.

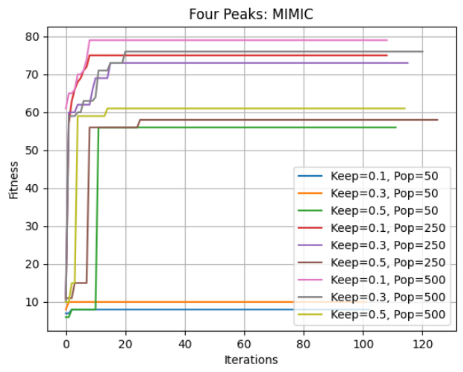

Figure 16 - MIMIC – Fitness vs Iterations

As it relates to the plot above, a population of 500 and keep percentage of 0.1(10%) are the ideal parameters from what was tested. This results from the fact that using a larger population size can lead to a more accurate probabilistic model, which can in turn lead to better candidate solutions. A higher keep parameter means that more candidate solutions are retained, which can maintain diversity in the population and prevent premature convergence to suboptimal solutions. The top performing parameter, as previously mentioned (keep = 0.1, population=500), creates a balance that allows this algorithm to find a solution quite quickly, however, it comes at the expense of computational resources and time.

5) *Algorithm Comparison*

It is difficult to say whether one algorithm is always better than other optimization algorithms for solving the Four Peaks problem, as the performance of each algorithm depends on a variety of factors such as the size and complexity of the problem, the specific parameters used, and the quality of the initial solutions.

Because of this, comparing the 4 algorithms is best done using an apples-to-apples approach. By comparing how each algorithm performs with respect to fitness vs problem size and iteration, it is easy to see which performs best. Figure 17 below does just this. On the left is fitness versus iterations, and on the right is fitness versus problem size.

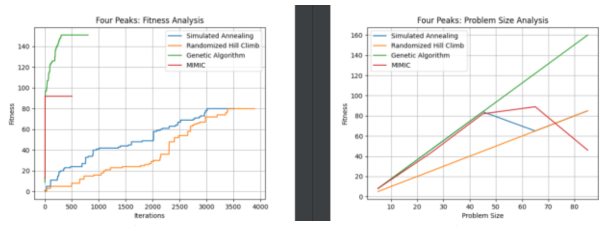

Figure 17 - Performance Analysis – Problem Size, left. Iterations, right

The plot on the right makes clear that GA, RHC, and SA perform similarly until the problem size grows to about 45. At that point, GA is the better algorithm. The plot on the left has a similar outcome. Here, SA and RHC are inferior solutions, leaving MIMIC and GA as the better of the two. Looking specifically at GA and MIMIC, it's clear that GA is slower at finding a solution when compared to MIMIC, however, the GA fitness score is much better.

Furthermore, the plot below comparing computation time of the 4 algorithms makes clear that MIMIC requires the most time to compute a solution, followed by GA, RHC and SA. Knowing that MIMIC and GA perform better than RHC and SA, the plot below shows very clearly at GA is the highlighted algorithm for the Four Peaks problem. This is supported by the fact that MIMIC takes about 4 times the amount of time (100 seconds) to find a solution when the problem size is 70, while GA takes about 25 seconds. Lastly, the GA time to compute is very linear, while MIMIC appears to lean more towards exponential growth. This means that as the problem size grows, it would be expected that GA would take a similar amount of time to solve, whereas MIMIC would take a much greater amount of time.

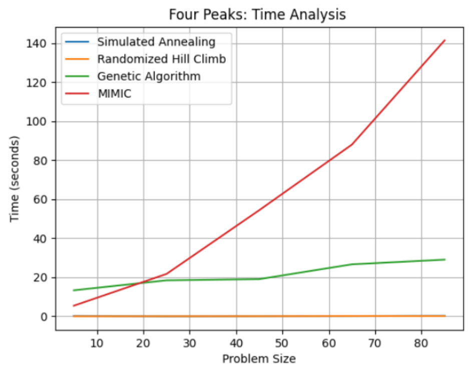

Figure 18 – Time Analysis

VI. Neural Network (NN) optimization

The objective of this section is to use the first three search algorithms to optimize weights for a neural network. The results will be compared to the backpropagation for the neural network used in Assignment 1.

To accomplish this task, the breast cancer dataset from Assignment 1 is used across all tests. To recap from Assignment 1, this dataset contains information on 569 patients and is composed of 357 benign cases (62.7% total) and 212 malignant cases (37.3% total). Each patient record in the dataset includes various features such as the patient's age, tumor size, and the presence of certain biomarkers, as well as a binary classification for diagnosis (benign or malignant). In total, each case has 30 features.

A. *Neural Network (NN) – Backpropogation*

As a baseline to compare with, a NN using gradient descent is implemented in mlrose. Figure 19 below shows the learning curve and loss curve.

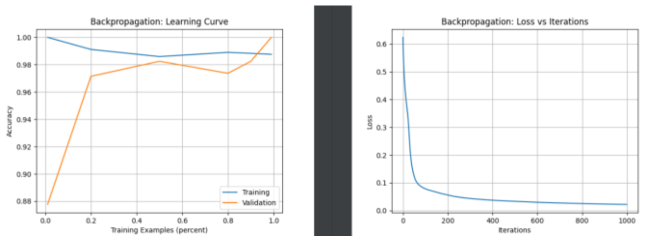

Figure 19 – Neural Network w/ Backpropagation – Breast Cancer Data Set. Learning Curve, left. Loss Curve, Right

The loss curve in figure 19 above (right image) shows a well-behaved loss curve with a decreasing trend in the loss function over time, without any sudden jumps or spikes. This displays a very high learning rate. On the left, a learning curve shows very effective learning on this dataset. With a maximum validation accuracy of about 98%, the learning curve converges after about 90% of the training examples are used. With little indication of overfitting, this is an ideal learning curve.

B. *Random Hill Climbing (RHC)*

As mentioned previously, RHC works by randomly generating an initial solution, evaluating its fitness by comparing it with its neighboring solutions, and replacing the current solution with it if it is better. As it relates to NN, both RHC and gradient boosting use similar principles, that is, an iterative approach that builds on itself based on its previous state.

The results of using an RHC algorithm to find optimal weights in the neural network are plotted below.

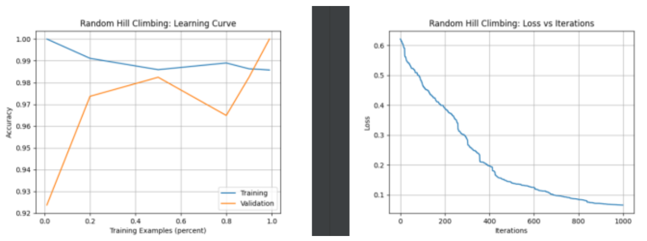

Figure 20 - Neural Network w/ RHC – Breast Cancer Data Set. Learning Curve, left. Loss Curve, Right

The loss curve, plotted on the right, for this neural network trained using RHC is very similar to the neural network loss curve using gradient boosting. However, the descent of loss takes more iterations, suggesting that the learning rate for this test is lower. Lastly, the learning curve for this NN using RHC looks very similar to the when using gradient descent, but accuracy for this test is higher.

C. *Simulated Annealing (SA)*

Compared to gradient descent, SA can be a more effective algorithm when optimizing a neural network because of its ability to better avoid local minima and finding the global minimum of the loss function. This is because SA allows the algorithm to explore a wider range of solutions, rather than just converging to the nearest minimum. Figure 21 below shows the learning curve, left, and the loss curve, right.

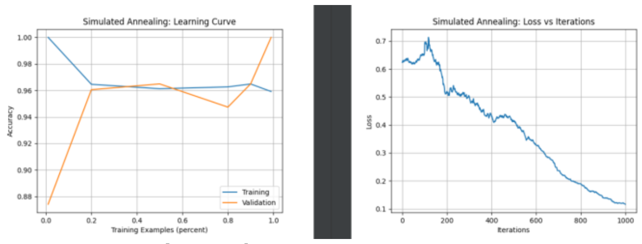

Figure 21 - Neural Network w/ SA – Breast Cancer Data Set. Learning Curve, left. Loss Curve, Right

Interestingly, the loss curve shows an increase in training loss within the first 100-150 iterations before it starts decreasing. Like the other learning curves previously discussed, training and cross-validation scores converge, but with SA this occurs much earlier at around the 40% of training examples used mark. Also, overfitting is a concern here, and can be observed in the learning curve starting around where 90% of the training examples are used.

D. *Genetic Algorithm (GA)*

Figure 22 below shows the learning curve and loss curve associated with optimization of a neural network using GA. Most noticeable is the sharp decline and recovery of the accuracy score in the learning curve. This could be the result of struggling to find a good solution but recovering after being optimized further. Also notable is the plateau in the loss curve. At between about 50 and 100 iteration, it is most likely that the GA optimization algorithm is stuck in a local optima, thus preventing it from further reducing its loss. After the 100th iteration, however, this GA algorithm continues and optimizes itself to find a good solution.

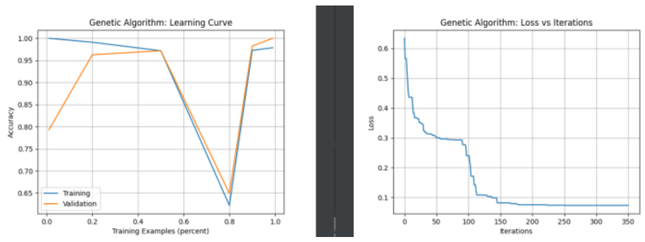

Figure 22 - Neural Network w/ GA – Breast Cancer Data Set. Learning Curve, left. Loss Curve, Right

E. *Comparative Analysis*

In this section, the best performing implementation of each algorithm, trained and tested with the breast cancer dataset, is compared. To accomplish this, each model's best training score and cross-validation score is reviewed. Lastly, training time is considered before determining the best algorithm used to optimize the neural network.

Looking at the plotted cross-validation and training scores analysis below in figure 23, it is clear that the neural network (NN) optimized using the simulated annealing algorithm has the lowest scores for both cross-validation and training data. The NN optimized using random hill climbing (RHC), genetic algorithm (GA), and backpropagation all score higher than SA, and take turns with which algorithm scores the highest on training and cross-validation scores. Backpropagation and RHC have the highest training score, but both backpropagation and RHC have lower cross-validation scores when compared to GA. As can be seen in the plot, the gap is relatively small though for both cross-validation and training scores. Between RHC, GA, and backpropagation, there is a score difference of only about 3%. With this information, GA did the best job of optimizing the NN as it has the highest cross-validation score, meaning it is the most effective when tested with unseen data and assessing its generalization ability. This GA algorithm will do well at avoiding overfitting and ensures that the model can generalize well to new data.

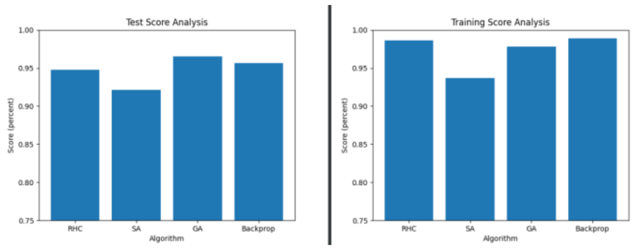

Figure 23 - Neural Network Performance Analysis Comparison – Breast Cancer Data Set. Cross-Validation Scores, left. Training Scores, Right

Looking at the time required to train each algorithm, there is a clear 'winner' and 'looser'. SA and backpropagation both require less than 5 seconds to train. GA is approximately 5 times slower than SA and backpropagation, requiring about 20 seconds to train. Lastly, RHC takes around 11 seconds to train.

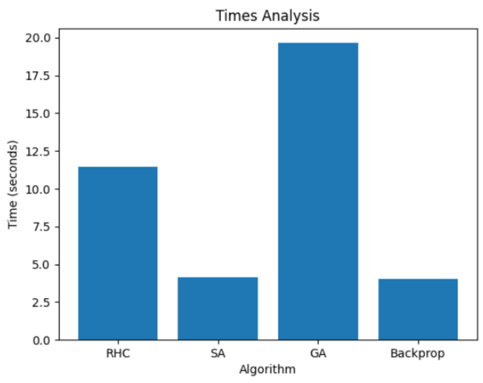

Figure 24 - Neural Network Time Analysis – Breast Cancer Data Set

With this information, the evaluation using only cross-validation and test scores is brought into question. While GA has slightly better scores, it takes significantly more time to train. The computational cost to train GA means that it might not actually be the 'winner'. As such, RHC actually has a greater test score than GA, and the cross-validation score is only about 2% lower at around 95%. With this information, the NN optimized using the RHC algorithm seems to be best choice for classifying the target variable in this breast cancer dataset.