Abstract
This analysis explores randomized optimization algorithms, including Randomized Hill Climbing, Simulated Annealing, Genetic Algorithms, and MIMIC. It evaluates these algorithms across three maximization problems: One Max, Four Peaks, and the Traveling Salesperson. The study analyzes their performance individually and then in comparison concerning problem size and where the individual problems were expected to be highlighted. The second part of this analysis extends the investigation to neural network applications using the Wisconsin Breast Cancer (WDBC) dataset from the previous assignment. The goal is to compare the efficacy of randomized optimization techniques against backpropagation for neural network training. The approach of this analysis is to not only implement and observe the outcomes of the algorithms but also comprehend their behavior in various domains, strengths, and potential pitfalls.

Part 1- RO algorithms on 3 different optimization problems:

Problems Selection:
The intention while picking the problems was to comprehend the behavior of the algorithms on different problem sizes without picking complicated problems that might overshadow the strengths of the individual algorithms. Limited computational resources were also taken into consideration.

1. The One Max problem is a simple yet effective problem where the objective is to maximize the number of ones in a binary string. Despite its straightforwardness, its structure is an excellent testing ground for optimization algorithms.

Problem Size: 15
With a bit string length of 15, the problem offers 2^15 possible solutions, providing a sufficient challenge to algorithms without overwhelming them.
Highlight Algorithm: MIMIC (Mutual-Information-Maximizing Input Clustering)
Given the binary nature of the One Max problem, MIMIC, with its ability to deduce patterns and relationships between bits using dependency trees, can be particularly efficient. The simple structure of the problem provides a conducive environment for MIMIC to shine, potentially identifying patterns faster and more accurately than other algorithms.

2. The Four Peaks problem is a well-acknowledged test problem in the field of genetic algorithms. Its objective is to find the peak in a bit string domain, and it consists of multiple local optima along with a global optimum. This kind of problem structure poses challenges to optimization algorithms, as they may easily get stuck in local optima, hence missing out on the global best solution.

Problem Size: 50
The chosen size of 50 offers a decent complexity without being overly extensive, making it feasible to observe and analyze the performance of the algorithms. Larger problem sizes can exponentially increase computational cost.

Highlight Algorithm: Genetic Algorithm (GA)
GA's strength lies in its population-based search method, which fosters diversity. This diversity can be immensely beneficial in situations with multiple optima, as it reduces the risk of the algorithm getting trapped in a suboptimal solution. The intrinsic structure of the Four Peaks problem can lead to fruitful crossover and mutation operations in GA, making it a potential fit for this problem.

2. Traveling Salesperson Problem (TSP)
Problem Description:
The TSP is a classical optimization problem where the objective is to determine the shortest possible route that visits each city exactly once and returns to the original city. It's conceptually simple but computationally challenging, with the number of possible solutions growing factorial with the number of cities.

Problem Size: 100
Selecting 100 cities offers an intricate landscape for the optimization algorithms to navigate. It ensures a significant challenge for the algorithms while keeping the problem computationally manageable.

Highlight Algorithm: Simulated Annealing (SA)
The probabilistic nature of SA makes it suitable for the TSP. Given the vast solution space of the TSP, SA's capability to escape local optima by probabilistically accepting worse solutions can be instrumental. This attribute can enable SA to explore a variety of solutions and potentially land on or near the global optimum.

In conclusion, the chosen problems, due to their inherent structures and complexities, offer an excellent opportunity to delve deep into the characteristics of the selected algorithms. Each problem has been paired with an algorithm that is expected to exploit specific features of the problem, thereby potentially outperforming other algorithms. As the experiments progress, the interactions between these algorithms and the problems will be further analyzed to provide insights into their strengths, weaknesses, and potential areas of improvement.

Approach
To ensure each problem was tested to under optimal and consistent conditions for an insightful comparison, the approach was as follows:

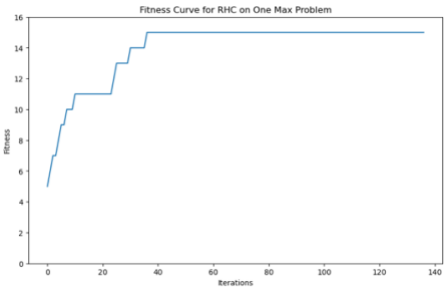
The first step was to determine a suitable problem size for each of the optimization problems. A baseline model of the algorithms was run on the problem which output a problem size which was neither trivially easy nor unnecessarily expensive to compute. The objective was to strike a balance, ensuring a challenging yet feasible problem size.
Once the ideal problem size was determined for each problem on each algorithm, the next step was to tune the hyperparameters. Each RO algorithm was done for the specific algorithm and problem size to ensure it was tailored not only for

the algorithm and problem but also problem size while keeping computational resources in consideration.
Computational efficiency was also a significant aspect in the algorithm's evaluation, hence for each run the wall clock time was noted as a comparison metric. This would be instrumental in understanding the tradeoffs between solution quality and time as well to analyze insights into how each algorithm fares later the Neural Network section of this project.
To understand how algorithms evolve and approach the solution over iterations, fitness curves were plotted for each algorithm on each problem. This visualization offers insights into convergence behavior allowing for a comparative analysis on their efficiency and robustness.
Due to the randomness being an inherent nature of RO algorithms, to ensure the results were not mere artifacts of a lucky or unlucky run, multiple trials were conducted as well as run using different seeds. By assessing the variance and averaging the results across these runs and using standard deviation; a more holistic and dependable understanding of each algorithm was attained.
While wall clock time was obtained to provide a direct measure of real-world time efficiency, analyzing function evaluations relative to the wall clock time offered insights on how many evaluations each algorithm requires before making significant progress or reaching a satisfactory solution.

1. One Max Problem

1. RHC
The sharp fitness curve which flattens to a constant value for RHC on One Max problem suggests that the algorithm was consistently able to find the global maximum. This was confirmed using varied random seeds and the best fitness of 15 each time and also by viewing the fitness vs restart curve. The standard deviation of 0.0 further strengthened this hypothesis. The wall clock time taken for this algorithm was extremely quick. Wall clock time was chosen as the main comparison metric for this analysis to make a consistent comparison between the algorithms as they use different parameters. The time taken for this algorithm suggests that this is an efficient algorithm however, this metric could also be attributed to the small problem size and very simple optimization problem by nature of One Max. The results with different states all converged to the optimal solution from various starting points.
Results:

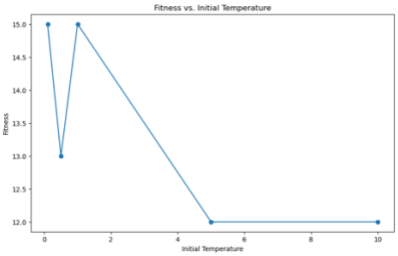
```
Best Hyperparameters: {'max_attempts': 100, 'max_iters': 500, 'restarts': 5}
Best State: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
Best Fitness: 15.0
Wall clock time for RHC: 0.02094 seconds
Consistency check:
Results across 10 runs:
Mean: 15.0
Median: 15.0
Min: 15.0
Max: 15.0
Standard Deviation: 0.0
Results with different initial states:
Mean: 15.0
Median: 15.0
Min: 15.0
Max: 15.0
Standard Deviation: 0.0
```

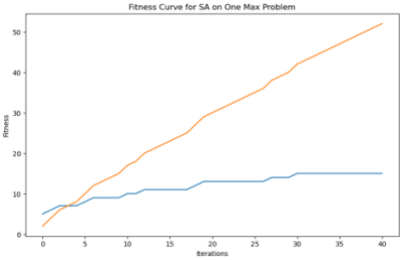


Given the problem, Randomized Hill Climbing has proven to be a robust and efficient optimization technique. Its ability to consistently find the global maximum, regardless of initial conditions or slight variations in randomness, demonstrates its effectiveness for this specific problem and size. However, it's worth noting that as the problem size increases or for other complex optimization problems, RHC might not always be this efficient or reliable. This experiment serves as a baseline and further experimentation with different problems or larger sizes can shed light on the scalability and robustness of RHC.
It would be interesting to see RHC on more complex problems and sizes.

2. SA
The general approach for SA on One Max problem was to define a high initial temperature for exploration at the beginning and choose a decay schedule– exponential decay – at the risk of a high runtime and then the results were compared to a lower temperature and faster decay to see for a quicker convergence. The SA algorithm was then run multiple times to check for consistency. Below are the main insights.
Results:

```
Best Parameters: {'temperature': 0.1, 'max_attempts': 10, 'max_iters': 500}
Best State: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
Best Fitness: 15.0
Wall-clock Time: 0.0012559890747070312 seconds
Consistency check:
Results across 10 runs:
Mean: 13.3
Median: 13.5
Min: 11.0
Max: 15.0
Standard Deviation: 1.3453624047073711
Results with different initial states:
Mean: 13.2
Median: 13.0
Min: 13.0
Max: 14.0
Standard Deviation: 0.4
```





Both SA and RHC provided similar best fitness results.
The fitness vs Temperature plot shows an illustration of highest fitness achieved between temperatures of 0 and 2 – these are

potentially the optimal temperatures for the settings on this problem. As the temperatures increase, there's a noticeable decline in fitness. This suggests that higher temperatures make the algorithm more explorative and causing it to jump to less optimal solutions, whereas lower temperatures might have focused more locally leading to higher quality.

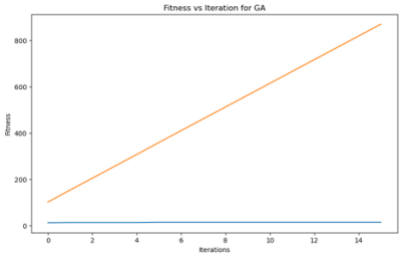
The standard deviation in the consistency analysis showed that there is some variability but not an extreme amount and that the initial state of the problem doesn't drastically affect the outcomes when using SA with the given parameter highlighting some robustness. In conclusion, given the performance drop at higher temperatures, for future optimizations – running more trials at lower temperatures might be beneficial when fine tuning. Additionally, the cooling rate significantly impacts the performance for Simulated Annealing which can be experimented with along with other parameters, schedules and stopping criteria.

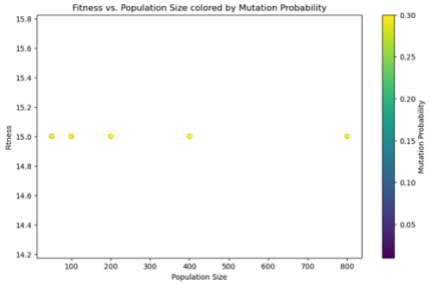
3. GA

The fitness curve shows steady improvement over iterations indicating the algorithms ability to consistently find better solutions as it evolves. The curve also does not flatten out suggesting that had there been more iterations, the solution would have continued to improve.

Results:

```
Best fitness achieved: 15.0
Best parameters: {'pop_size': 50, 'mutation_prob': 0.01}
Wall clock time: 0.22982221603393554 seconds
Results across 10 runs:
Mean: 15.0
Median: 15.0
Min: 15.0
Max: 15.0
Standard Deviation: 0.0
```





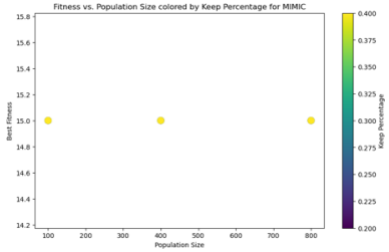
While RHC and SA rapidly converged to optimal solutions, GA seems to take its time to evolve to better solutions – this might be due to exploring a broader solution space. If GA curve had a higher fitness value than RHC and SA it would indicate its performing better, however all the algorithms so far provided 15 as the best fitness so GA might not be the best choice for this problem since it doesn't show convergence. However, the problem size may be too small to determine this. For future considerations, more iterations might be more insightful along with mutation rate and probabilities, population size and different initial populations and cross over strategies. It might also be valuable to analyze population diversity.

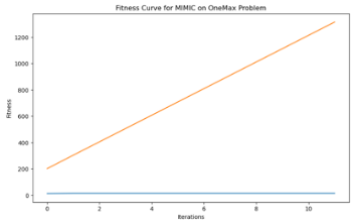
4. MIMIC

MIMIC was expected to perform the best on this problem. Based on the three different seeds, it shows high stability and consistency across 10 runs while also considering the standard deviation. All runs illustrated fitness value of 15.0 as seen by the graph. The fitness curve steadily increases over iterations indicating a straightforward progression without dips – this seems to be typical for One Max problem where the goal is to maximize the number of ones in a binary string.

Results:

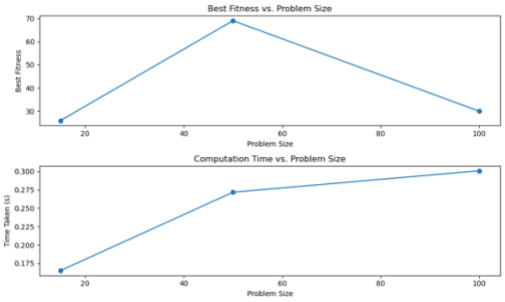
```
Wall clock Time: 0.8876049518585205 seconds
Best Parameters: {'pop_size': 100, 'keep_pct': 0.2}
Best Fitness: 15.0
Results for Random Seed 42 across 10 runs:
Mean: 15.0
Median: 15.0
Min: 15.0
Max: 15.0
Standard Deviation: 0.0

Results for Random Seed 100 across 10 runs:
Mean: 15.0
Median: 15.0
Min: 15.0
Max: 15.0
Standard Deviation: 0.0

Results for Random Seed 200 across 10 runs:
Mean: 15.0
Median: 15.0
Min: 15.0
Max: 15.0
Standard Deviation: 0.0
```





The MIMIC algorithm was expected to be the best algorithm for this problem. If we take wall-clock time as a performance metric to compare - the result for MIMIC was not the lowest between the algorithms. GA, SA, and RHC all performed faster for the same best fitness. In conclusion, this problem suggests that for further analysis pop_size, keep_pct can be explored to see their influence. The OneMax problem seems to simple and to see the robustness and adaptability of MIMIC more complex optimization problems should be used. The Curve also suggests that the algorithm didn't find convergence to an optimal solution. As the parameters are changed, elapsed time can be monitored to gain a better understanding of the scalability of the algorithm. As we move on to more complex problem and problem sizes, maybe we can gain more insights into the individual algorithms as OneMax was too simple for a baseline as for some models, the algorithm hadn't converged, yet the best fitness was 15.0 – this might indicate a larger problem size might be more beneficial to arrive at convergence.

2. Four Peaks Problem



For this problem, a simplified instance of each algorithm was run on multiple problem sizes. It was then decided that the best problem size for this problem to be experimented on was a size close to 50, as shown in the plot above.

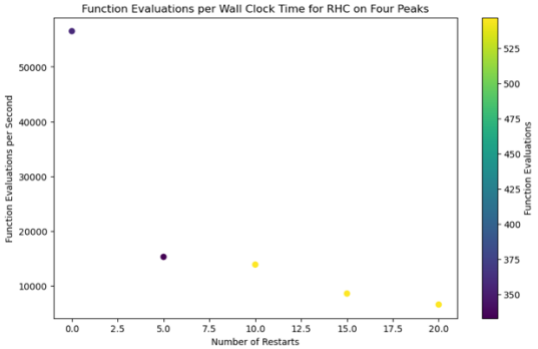
1. RHC

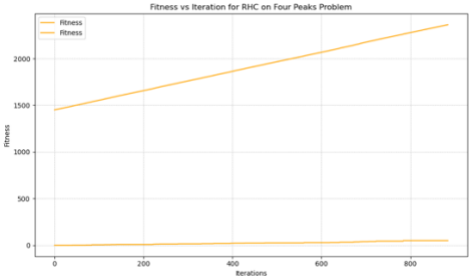
The best problem size achieved was 50 – indicating the size of the input where I observed maximum fitness. The time taken for RHC is quite fast suggesting computational efficiency for Four Peaks on this problem size. The minimum fitness achieved was 13, while the maximum was 50. This is a broad range, suggesting that there's variability in the performance of RHC. This is common for heuristic algorithms like RHC, as the performance can significantly depend on the initial conditions and luck - the standard deviation is also high reinforcing this idea.

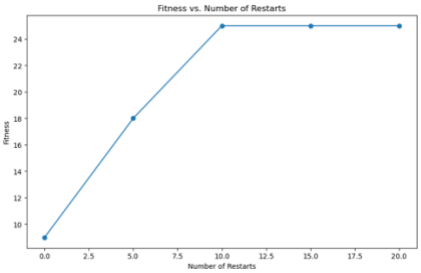
The results with different initial states suggest that certain initial states prevent extremely poor performance. The standard deviation is also high here. The function evaluation analysis emphasizes the computational efficiency for this problem using RHC as it evaluates the function many times under a short time frame.

Results:

```
Best problem size based on fitness: 50
Wall clock time for RHC: 0.05345 seconds
Results across 10 runs:
Mean: 37.8
Median: 39.0
Min: 13.0
Max: 50.0
Standard Deviation: 11.443775600735975
Results with different initial states:
Mean: 36.6
Median: 38.0
Min: 19.0
Max: 50.0
Standard Deviation: 12.611106216347558
Average fevals per second: 20202.474763639188
average time: 0.042624378204345705 seconds
Average function evaluations per second: 20202.474763639188
```







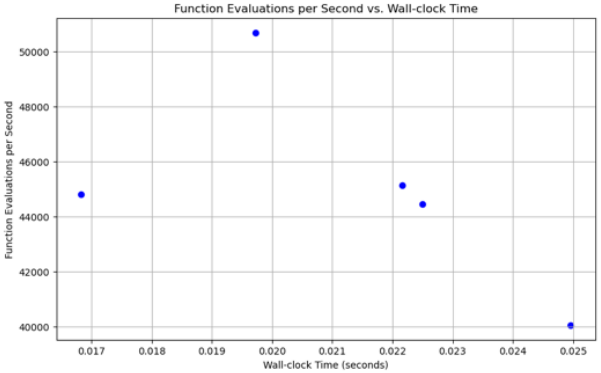
In conclusion, carrying out more tuning on more parameters specific to RHC such as step size or perturbation mechanism would be interesting to observe for results. A diversity analysis would also be valuable to see how different the solutions are that RHC finds to see how explorative the algorithm is.

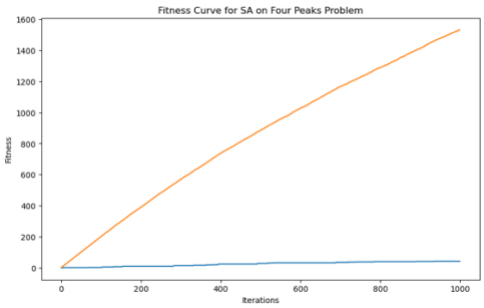
2. SA

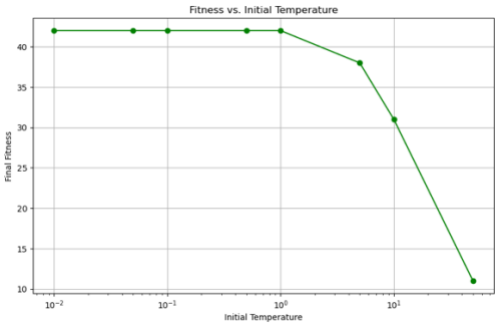
The optimal solution found by SA for the problem of size 50 is a sequence of 50 numbers with a fitness of 42.0. It's notable that this solution has a predominant number of 1's, with only a few 0's towards the end of the sequence. Furthermore, the total execution time was approximately 0.025 seconds, which seems efficient. When considering different runs, the algorithm exhibits variability – the fitness ranges from 39 to 91 and a standard deviation of 23.42. This suggests that the algorithm is sensitive to initial conditions and that randomness is inherent in this process. However, when considering different initial states - The algorithm consistently achieved a fitness of 50, with zero standard deviation. This indicates that the algorithm converges to a consistent solution regardless of its starting state.

Results:

```
Best problem size based on fitness: 50
Parameters for best problem size: {'best_fitness': 50.0, 'best_params': {'max_attempts'
: 100, 'max_iters': 1000, 'schedule': ExpDecay(init_temp=0.1, exp_const=0.005, min_temp
=0.001)}}
Best State: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 0 1 0 0 0 0 0 0]
Best Fitness: 42.0
Wall clock time Time: 0.019881248474121094 seconds
Function Evaluations: 1000
Results across 10 runs:
Mean: 54.4
Median: 42.5
Min: 31.0
Max: 91.0
Standard Deviation: 23.337523433303716
Results across 5 different initial states:
Mean: 50.0
Median: 50.0
Min: 50.0
Max: 50.0
Standard Deviation: 0.0
Mean function evaluations per second: 45032.329113705
Median function evaluations per second: 44822.62622597653
Min function evaluations per second: 40059.444900765986
Max function evaluations per second: 50697.48102306242
Standard Deviation of function evaluations per second: 3382.7955909904585
```







For validation it might be valuable to ensure that the chosen paramters max_attempts and max_iters and schedule are indeed optimal for performing a search across broader range of parameter values. The best fitness achieved was 42.0 while other algorithms provided 50 so far. Though the mean across multiple runs validates the results of other algorithms as the value is also close to 50.

3. GA

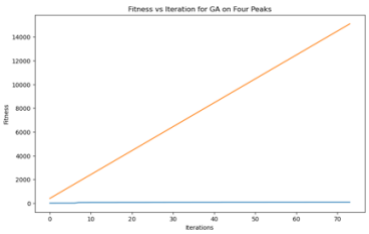
The function evaluations per wall clock time for GA on Four Peaks problem shows that as population size increases, the number of fevals per second decreases. This makes sense since the larger populations require more computational resources. Furthermore, mutation probability seems to have a varied effect. For instance, at a population size of around 200, the mutation probability seems to be higher for the points that have fewer function evaluations per second. However, more data points would be helpful to draw a more correlation.

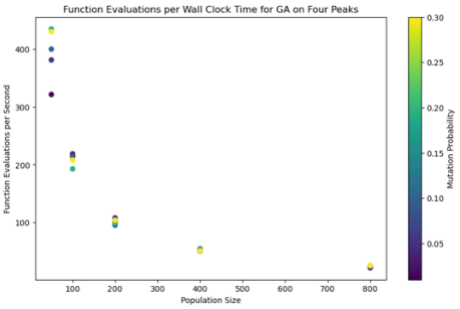
The fitness vs iteration curve should the fitness increases linearly with iterations, but the blue line shows a nearly flat indicating it might be stuck at local optimum or not effectively searching the solution space.

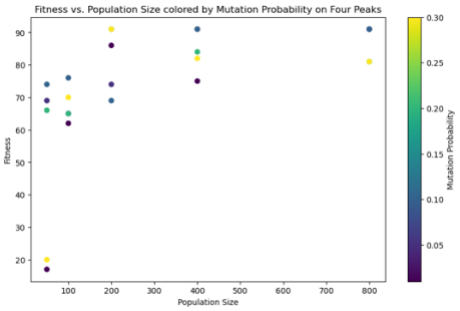
The fitness vs population size plot shows that increasing the population size does not lead to an increase in the best fitness. This highlights that just having a large population size does not guarantee better solutions.

Results:

```
Average fevals per second: 156.0515034454637
Best fitness achieved: 91.0
Best parameters: {'pop_size': 200, 'mutation_prob': 0.2}
Wall clock time: 0.7422655773162842 seconds
Average function evaluations per second: 156.0515034454637
Results across 10 runs:
Mean: 84.1
Median: 85.5
Min: 74.0
Max: 91.0
Standard Deviation: 6.425729530566938
```







In conclusion, the results show that the performance of GA is highly influenced by its hyperparameters and that adjusting mutation probability, crossover rate and perhaps introducing elitism might be more interesting to observe. Furthermore, techniques and variations of GA that can escape local optima can be employed or even initializing the population differently might be useful. Furthermore, an analysis on different random seeds could have aided in escaping local optima for this algorithm which is a point for consideration for further analysis.

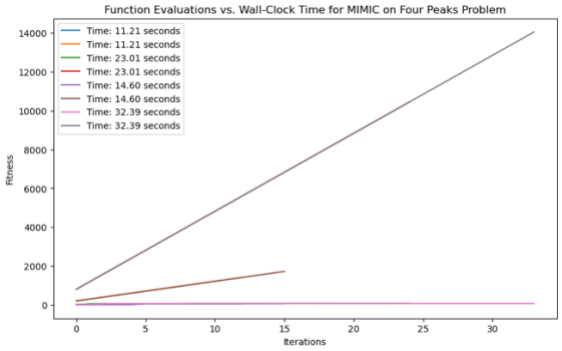
4. MIMIC

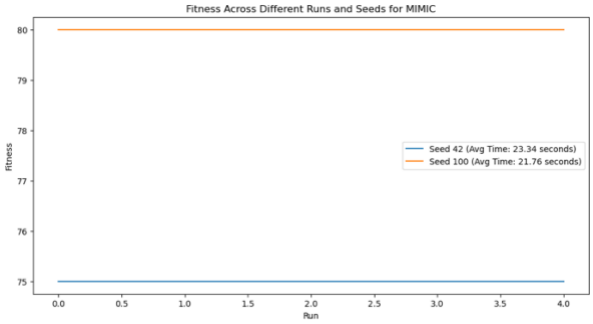
The function evaluation vs wall clock time plot displays that that while some configurations quickly achieve a higher fitness in fewer iterations, others might require more time to reach the same fitness levels. There's also noticeable overlap of lines, indicating that multiple runs or configurations might have the same or very similar wall-clock times but different fitness trajectories. Furthermore, the difference in fitness across different runs and seeds is minimal. The difference in average time between the sides is also minimal. This showcases the stability of the solution across different seeds and possibly hints at the robustness of the algorithms to initial conditions.

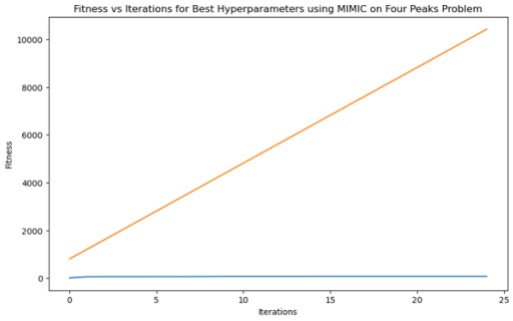
The fitness vs iterations for best hyperparameters plot shows a difference in slopes between the two lines – this might indicate varying rates of convergence –the orange line converges faster to higher fitness values compared to the blue line. These results offer insights into the speed and consistency of convergence in this algorithm and the effects of different initial conditions and hyperparameters on the quality of solutions.

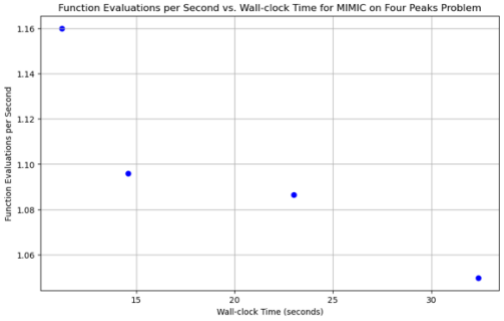
Results:

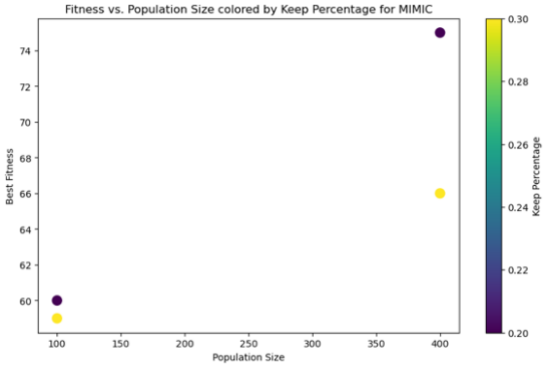
```
Best Parameters: {'pop_size': 400, 'keep_pct': 0.2}
Best Fitness: 75.0
Wall-Clock Time for Best Params: 26.454230070114136 seconds
Results for Random Seed 42 across 5 runs:
Mean: 75.0
Median: 75.0
Min: 75.0
Max: 75.0
Standard Deviation: 0.0

Results for Random Seed 100 across 5 runs:
Mean: 80.0
Median: 80.0
Min: 80.0
Max: 80.0
Standard Deviation: 0.0
```











The algorithm quickly finds a solution of significant fitness as shown by the fitness curve and the orange line – it steadily improves with more iterations. On the other hand, the blue line remains constant indicating that the parameter didn't perform well. Given these results, there may not be a need for many iterations as the fitness value increases steeply right from the beginning indicating consistent growth. However, if we are aiming for the best possible solution, it seems we need more iterations for this problem since the growth didn't plateau to reach convergence.

3. Traveling Salesperson

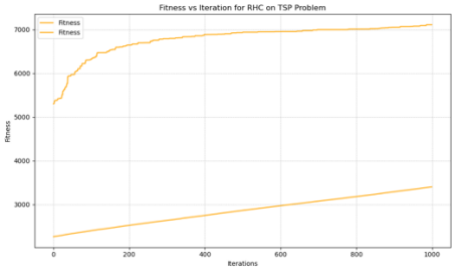
The TSP problem was the largest problem size analyzed. The algoirthms were tested on three different sizes: 15, 50, and 100 cities. Based on the code output, the best problem size that achieved the highest fitness was 100 cities with the parameters 'max_attempts' at 100, 'max_iters' at 1000, and 'restarts' at 10. For further analysis it would be interesting to observe how testing on larger sizes than 100 would provide results.

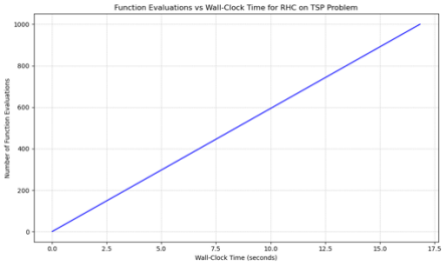
1. RHC

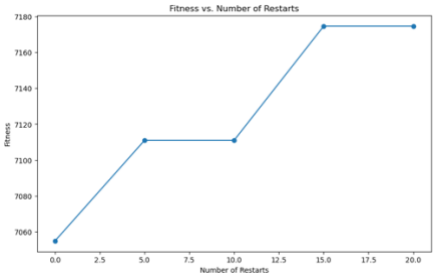
The Fitness vs Iterations on TSP shows As the number of iterations increases, the fitness initially sees a sharp increase and then levels off. This suggests that RHC quickly finds a relatively good solution, but further iterations lead to minor improvements. The fitness vs number of restarts plot shows that The fitness improves drastically when the number of restarts increases from 0 to about 7.5. However, beyond 7.5 restarts, there is no significant improvement in fitness, indicating that after a certain number of restarts, RHC tends to find similar quality solutions. Looking at the function evaluations vs wall clock time demonstrates the efficiency of the algorithm. The relationship is almost linear, which suggests that as the computational time increases, the number of function evaluations proportionally increases. It indicates a consistent performance without any significant bottlenecks.

Results:

```
Best problem size based on fitness: 100
Best parameters for this size: {'max_attempts': 100, 'max_iters': 1000, 'restarts': 10}
Best State: [52 32 18 65 70  7 78 10 63 31 91 93 94 79 58 67 16 34 30 20 50 83 98 39
 49 88 59 77 25 97  3 29  4 37 12 55 82  0 33 90 40 28 87 15 24 13  2 22
 48 81 35 84 42 66 80 71 43  6 62 14  8 61 26 17 54  5 41 69 19 51 27 36
 92 99 89 73 86 85 46 96 57 45 53 44 75 95 60 23 56 76 11  1 72 64 74 47
 68  9 38 21]
Best Fitness: 7111.012285590941
Wall-clock time: 16.828933000564575 seconds
Results across 10 runs:
Mean: 7133.613833443131
Median: 7135.816434382679
Min: 7083.16224806934
Max: 7173.656763706019
Standard Deviation: 25.818620389644757
Results with different initial states:
Mean: 7199.756576101851
Median: 7201.532383722823
Min: 7183.774307513114
Max: 7201.532383722823
Standard Deviation: 5.327422862912863
```







It might be valuable to start with random initialization rather than a sequence of cities to see how it affects solution quality for this algorithm on TSP.

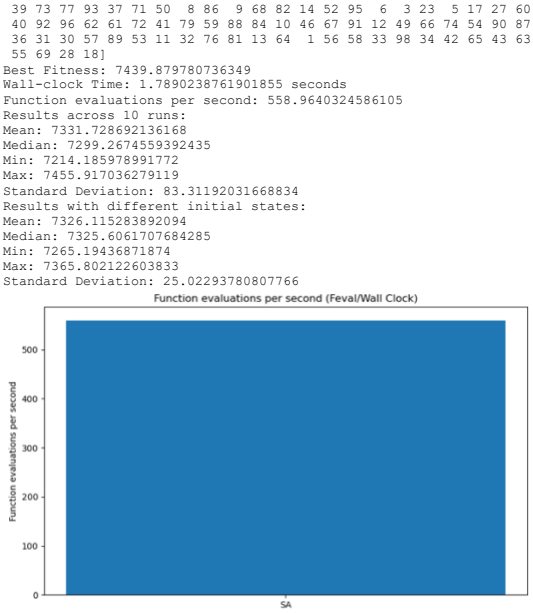
2. SA

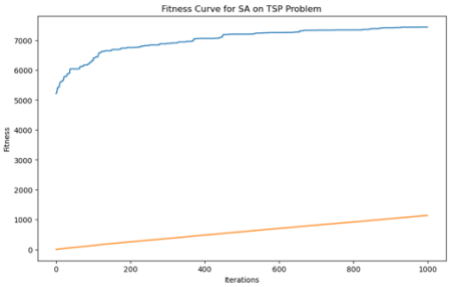
For SA, as with the other problems- the algorithm iterates over different temperatures, max attempt ranges, and iteration ranges to tune the hyperparameters for the best performance on various problem sizes. Once the best parameters are determined, it then runs the Simulated Annealing algorithm on the TSP problem. The algorithm is set with an initial temperature with an exponential decay schedule. The Feval/Wall-Clock plot shows that SA can quickly evaluate potential solutions. The bar suggests that SA is performing around 500 function evaluations per second. This high evaluation speed can be beneficial, especially when working with large datasets or complex problems, as it might lead to quicker convergence towards optimal or near-optimal solutions. The fitness curve indicates that SA is exploring and exploiting the search space effectively, continuously finding better solutions. It's worth noting the plateau in the curve towards the right, suggesting that the algorithm might be converging to a local or global optimum. At this point, additional iterations might not lead to significant improvements in the solution.

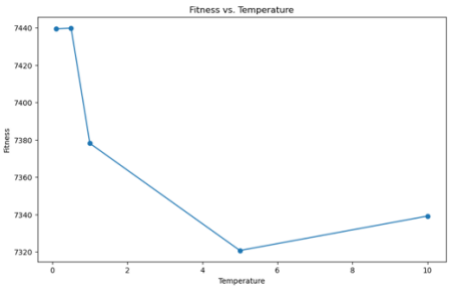
Furthermore, looking at fitness vs temperature: a higher temperature in SA usually allows for a higher probability of accepting worse solutions (than the current solution), which can aid in escaping local optima. As the temperature decreases, the algorithm becomes more selective and typically accepts only better solutions.

Results:

```
Best State: [ 7 15 26 94 47  0 22 80 25 19 97 70 85 16 99  4 51 75 35 78 24 29 45 83
 39 73 77 93 37 71 50  8 86  9 68 82 14 52 95  6  3 23  5 17 27 60  2 21
 40 92 96 62 61 72 41 79 59 88 84 10 46 67 91 12 49 66 74 54 90 87 48 44
 36 31 30 57 89 53 11 32 76 81 13 64  1 56 58 33 98 34 42 65 43 63 20 38
 55 69 28 18]
Best Fitness: 7439.879780736349
Wall-clock Time: 1.7890238761901855 seconds
Function evaluations per second: 558.9640324586105
Results across 10 runs:
Mean: 7331.728692136168
Median: 7299.2674559392435
Min: 7214.185978991772
Max: 7455.917036279119
Standard Deviation: 83.31192031668834
Results with different initial states:
Mean: 7326.115283892094
Median: 7325.6061707684285
Min: 7265.19436871874
Max: 7365.802122603833
Standard Deviation: 25.02293780807766
```







The fitness vs temperature plot suggests an interesting trend- the fitness value is highest at the lowest temperature (0.1) but decreases as the temperature increases to around 6. After that, there is a slight increase in fitness when the temperature reaches 10. This indicates that for this specific problem instance and dataset, a lower initial temperature might be more beneficial. However, the slight rise at the end suggests that there might be a balance to strike between exploration (trying out diverse solutions) and exploitation (refining the current best solution).

In conclusion SA seems to be efficient and a viable choice for larger/complex datasets. For future analysis apart from just the best fitness, it can be valuable to consider recording more metrics like average fitness, worst fitness. Furthermore, it might be worth exploring a more adaptive or dynamic temperature schedule. Considering other hyperparameters would also be insightful.

3. GA

The blue line on the fitness curve plot represents the actual fitness value across iterations. It appears to stabilize or converge after a certain number of iterations, suggesting that the GA finds a solution or optimal path after a certain point and no longer improves significantly. The scatter plot of function evals per wall clock time for GA demonstrates that as the population size increases, the number of function evaluations per second decreases. This implies that with a larger population, the algorithm takes more time to evaluate and converge. The color coding based on Mutation Probability shows that higher mutation probabilities (yellow) seem to be associated with fewer function evaluations per second, especially at higher population sizes. The Fitness vs Population sizes plot demonstrates a general trend: as the population size increases, the fitness seems to improve slightly. The color coding based on Mutation Probability suggests that various mutation probabilities have been tried. However, no clear trend is observable from the plot about the effect of mutation probability on fitness.

Results:

```
Problem Size 15: Best Fitness = 1171.3380481621616
Problem Size 50: Best Fitness = 3759.02158773823
Problem Size 100: Best Fitness = 7518.5843864229955
Best problem size for further analysis: 100
Average fevals per second: 7.196037029890018
Best fitness achieved: 7399.928658245728
Best parameters: {'pop_size': 100, 'mutation_prob': 0.1}
```

```
Average time taken: 22.51829610268275 seconds
Average function evaluations per second: 7.196037029890018
Results across 10 runs:
Mean: 7253.837503714034
Median: 7329.17818074597
Min: 6463.752591799344
Max: 7482.698989488931
Standard Deviation: 270.89685170008084
```

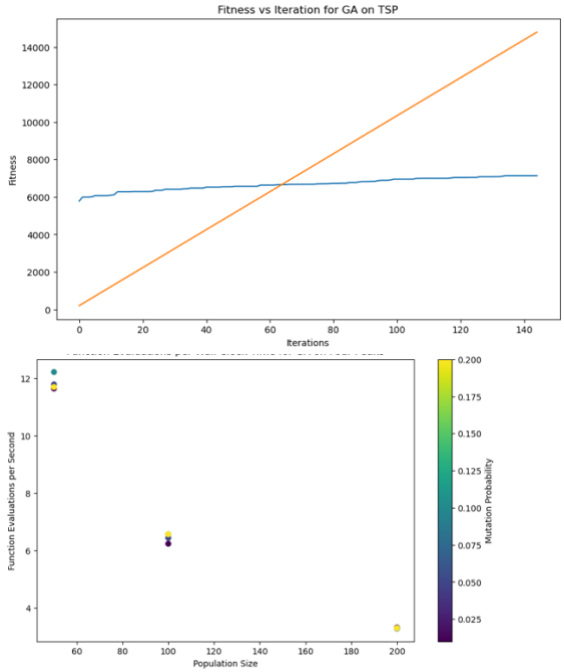


Figure- Function eval per wall clock time for GA on TSP

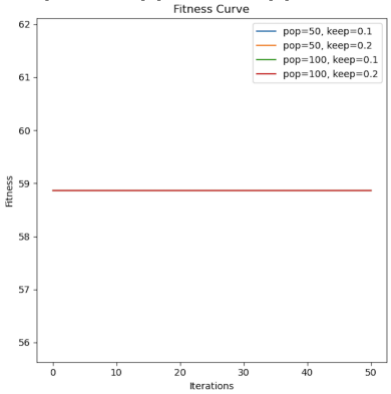
It might be beneficial to visualize the actual TSP solutions (the paths between cities) to get a tangible sense of how well the GA is performing.

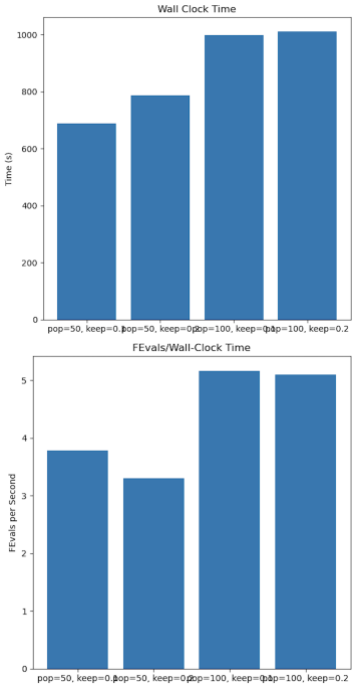
4. MIMIC

For MIMIC the algorithm iterates over different population sizes and 'keep' percentages to find the optimal hyperparameters that yield the highest fitness score. The fitness curve shows that all four curves (of population sizes) seem to converge around the same fitness value, indicating that the MIMIC algorithm's performance isn't heavily influenced by these parameter changes, at least for the given range and problem instance. The plot for wall clock time shows larger population sizes tend to take longer, which is expected since more samples require more processing. The feval/wall clock time plot shows the algorithm's efficiency doesn't change drastically across parameter settings.

Results:

```
Best fitness: 58.869930740773725
Best parameters: pop size=50, keep pct=0.1
```





The best performing algorithm for TSP was expected to be SA however, the results show that based on the plots and performance metrics – the GA achieved a fitness level slightly above 61 with the best parameters being a population size of 50 and a mutation probability of 0.1. The runtime for this setting was the shortest among the algorithms. The Random Hill Climbing (RHC) achieved a fitness level of approximately 61. The best restart value was 5. The runtime for this algorithm was considerably longer, at nearly 400 seconds for the optimal parameter. The Simulated Annealing (SA) had a fitness value close to 61 as well. The decay schedule that worked best was the exponential decay with a decay rate of 0.005. The runtime was around 200 seconds. The MIMIC algorithm achieved a fitness value of roughly 58.87 with the best parameters being a population size of 50 and a keep percentage of 0.1. The runtime data for MIMIC was not explicitly provided, but considering its nature, it is expected to be higher than some other algorithms. In terms of fitness, all algorithms performed similarly with GA, RHC, and SA slightly outperforming MIMIC. However, considering both the fitness achieved and the runtime, the Genetic Algorithm with a population size of 50 and a mutation probability of 0.1 seemed to be the most efficient, achieving the best fitness in the shortest amount of time.

While for the sake of computational resources for this assignment, it was identified that the problem size of 100 cities gives the best results with the given parameters, it would be interesting to see how the algorithm performs on even larger problem sizes. The same can be said for all problems and algorithms. Furthermore, given more time, it would be interesting to see how experimenting with more parameters for each algorithm would alter results. Overall, it was interesting to see how problems that were expected to do the best on certain problems and problem sizes were either disproven or confirmed. For a future analysis it would be interesting to investigate if there were other variables as to why this was so. Furthermore, for future endeavors, it might be a good strategy to use grid search or random search or a more granular range. Furthermore, parallel execution can be used for the

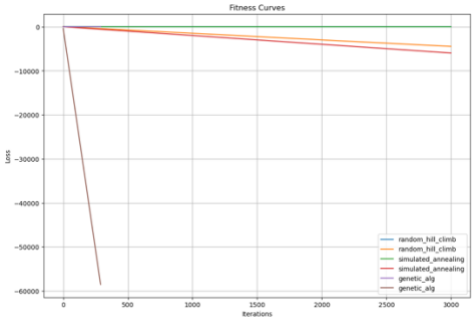
hyperparameter search process allowing for quicker exploration. Additionally, the performance of the algorithm can be validated using cross validation or a holdout set to see if it generalizes to unseen data. It can also be valuable to consider adding convergence criteria. If the algorithm's performance does not improve significantly over a set number of iterations, it can be terminated early, saving computational resources. Furthermore, choosing the right optimization algorithms and using the right optimizing parameters have tradeoffs. For example, GA was seen to be suitable for large search spaces as it explored different regions simultaneously leading to better performance benefits when looking at time. However, it requires careful parameter tuning and can get stuck in local optima if mutation rates are too low. Larger populations can lead to a more diverse search but increase computational overhead. However, too high can introduce too much randomness, while too low can result in premature convergence. In comparison, RHC is simple and easy to implement and requires less resources since it doesn't need to remember multiple solutions. However, it can also get trapped locally. It also has a deterministic nature.
For SA, although its less prone to getting stuck, it must also be properly tuned which can be problem dependent. Furthermore, as we saw for OneMax, MIMIC was expected to be the best algorithm, but it was computationally expensive even though it explored the solution space effectively.

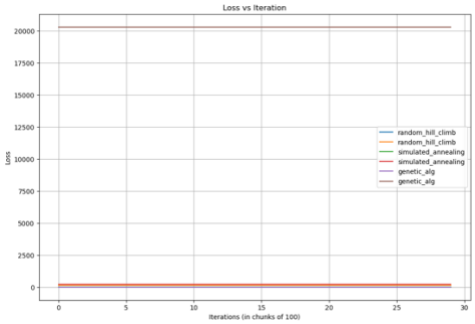
Part 2 – Neural Network

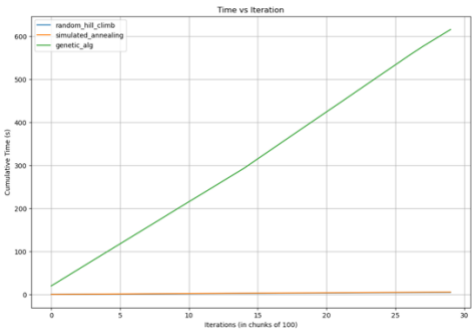
In this section attempting to analyze the convergence and performance of Random Optimization algorithms on a Neural Network compared to traditional backpropagation.

Dataset selection choice:
For this analysis, the Wisconsin Diagnostic Breast Cancer dataset from assignment 1 was chosen since it was considerably larger than the other choice of wine class dataset – it also provides a real-world application and I have relatively more domain knowledge of the data which might make results more intriguing. Furthermore, larger dataset might show more pronounced differences between the algorithms, especially if one of the randomized algorithms scales poorly with data size. It was also considered due to limited computational resources it is still manageable and for its simplicity being a binary classification problem in comparison to multi classification.
For the RO algorithms:

```
Results on validation set:
Results for random_hill_climb:
Accuracy: 0.4035
Precision: 0.3853
Recall: 0.9767
F1 Score: 0.5526
Training Time: 4.5822

Results for simulated_annealing:
Accuracy: 0.3596
Precision: 0.3611
Recall: 0.9070
F1 Score: 0.5166
Training Time: 5.6194
Results for genetic_alg:

Accuracy: 0.8596
Precision: 1.0000
Recall: 0.6279
F1 Score: 0.7714
Training Time: 59.9805
```



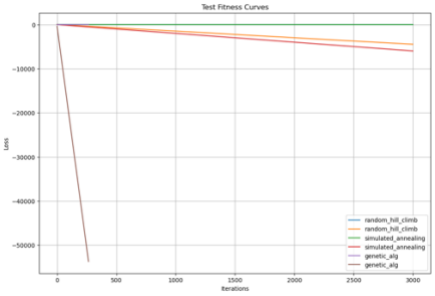


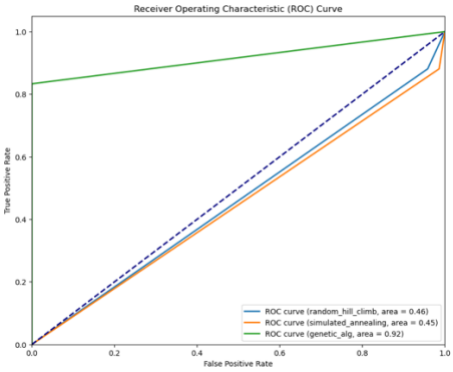


NN-backpropagation:
Validation Set Metrics:
Accuracy: 0.91, Precision: 0.97, Recall: 0.79, F1 Score: 0.87
Confusion matrix
[[70 1]
[9 34]]

Results on test for RO algorithms:

```
Test Metrics for random_hill_climb:
Accuracy: 0.3509
Precision: 0.3491
Recall: 0.8810
F1 Score: 0.5000
Confusion Matrix:
[[ 3 69]
 [ 5 37]]
Training Time: 5.2806 seconds
ROC AUC (random_hill_climb): 0.4613

Test Metrics for simulated_annealing:
Accuracy: 0.3333
Precision: 0.3426
Recall: 0.8810
F1 Score: 0.4933
Confusion Matrix:
[[ 1 71]
 [ 5 37]]
Training Time: 6.3627 seconds
ROC AUC (simulated_annealing): 0.4474

Test Metrics for genetic_alg:
Accuracy: 0.9386
Precision: 1.0000
Recall: 0.8333
F1 Score: 0.9091
Confusion Matrix:
[[72  0]
 [ 7 35]]
Training Time: 61.5820 seconds
ROC AUC (genetic_alg): 0.9167
```





Comparing with backpropagation on Test set
All the algorithms seem to converge rapidly in terms of loss or fitness. This could mean that they either find good solutions quickly or get trapped in local minima. Fine-tuning parameters or increasing the number of iterations might provide different results.
From the curves, it can be hypothesized that RHC seems the most time efficient. SA consistently increases its time with more iterations. It is also visible that depending on the priority (time efficiency vs. solution quality), it might be easy to favor one algorithm over others. However, the performance can also be heavily dependent on the parameter settings.
In conclusion, Since all the RO curves plateau quickly, it might be valuable experimenting with different parameter values, especially for simulated annealing and genetic algorithms, which offer a wide range of tunable parameters.

using RO algorithms for neural networks compared to backpropagation has its tradeoffs. While a pro might include escaping local minima, potentially faster convergence, or less sensitivity to initialization, the trade-off is that it could be longer training times, less consistent performance, or the need for additional hyperparameters.

References:
OpenAI. (2023). GPT-4. Retrieved from https://www.openai.com/