PART 1

Welcome to our exciting journey into the world of machine learning and optimization algorithms! Our travel companions on this adventure are the dynamic quartet of Random Hill Climbing (RHC), Simulated Annealing (SA), Genetic Algorithms (GA), and MIMIC.

Our goal is to understand three unique optimization problems, each with its own set of challenges. I picked 2 maximization problems, four peaks and knapsack and 1 minimization problem, the Travelling Salesman Problem.

Next we will dive into the realm of neural networks, using our algorithms to optimize weights and enhance performance. It's a competition with backpropagation, the traditional technique in the field.

The reason for my selection of the optimization problems are as such,
I chose the 4 peaks problem as it is interesting because it has both a global maximum and a deceptive local maximum, which can trap simpler algorithms like RHC. This problem highlights the strength of GA, which can avoid getting stuck in local maxima by combining fit sub-pieces of lower-fitness members of the population.

The next problem was the Knapsack Problem, as it is a combinatorial optimization problem. It's interesting because it requires finding the best combination of items given a weight constraint, which can be quite challenging. This problem can highlight the strengths of SA, which uses a probabilistic technique to escape local optima and has the potential to find the global maxima.

The next problem I chose was the traveling salesman problem. I chose this problem as it is one of the very famous NP Hard problems in computer science and also has a lot of practical applications as it optimizes the minimum distance you need to cover in order to traverse the needed locations.

For each problem, we've carefully defined a fitness function and created a problem instance. We've also set up a runner for each algorithm on each problem with specific parameters.

For RHC, we've set the maximum attempts to 500 and used a range of restarts. This allows the algorithm to escape local optima by restarting the search process from a different randomly selected point.

For SA, we've experimented with different temperature schedules (0.1, 0.5, 0.75, 1.0, 2.0, 5.0) using geometric decay. This allows the algorithm to control the balance between exploration i.e. searching new areas and exploitation i.e. searching around the current best area.

For GA, we've set the population size to 200 and mutation rate to 0.1. These parameters control the diversity of solutions in the population and the rate at which individuals in the population mutate, respectively.
For MIMIC, we've set the population size to 200 and keep percent to 0.2. These parameters control the number of samples in each generation and the proportion of samples to use as parents for generating the next generation.

We've also performed hyperparameter tuning to optimize these parameters for each algorithm on each problem. This is a crucial step as it can significantly improve the performance of our algorithms.

In terms of running the algorithms, we've used an iteration list which allows us to see how the performance changes with increasing iterations.

Analysis

We'll start by presenting the top 20 runs for each algorithm on each problem. These are the runs that achieved the highest fitness scores. By examining these top-performing runs, we can gain insights into what makes a run successful. We'll present these results in a table format for easy comparison. Then following it we would highlight the single best run for each algorithm on each problem. This is the run that not only achieved the highest fitness score but also did so with the minimum number of function evaluations. This run represents the most efficient solution found by the algorithm.

Results for RHC on Four Peaks Problem:

	Iteration	Time	Fitness	FEvals	Restarts	max_iters	current_restart
0	0	0.000202	1.0	0.0	0	1024	0
1	1	0.001603	1.0	1.0	0	1024	0
2	2	0.002577	1.0	2.0	0	1024	0
3	3	0.003571	1.0	3.0	0	1024	0
4	4	0.003598	1.0	4.0	0	1024	0
5	5	0.006324	1.0	5.0	0	1024	0
6	6	0.006358	1.0	6.0	0	1024	0
7	7	0.006379	1.0	7.0	0	1024	0
8	8	0.006399	1.0	8.0	0	1024	0
9	9	0.008721	1.0	9.0	0	1024	0
10	10	0.008757	1.0	10.0	0	1024	0
11	11	0.008780	1.0	11.0	0	1024	0
12	12	0.008800	1.0	12.0	0	1024	0
13	13	0.008819	1.0	13.0	0	1024	0
14	14	0.008838	1.0	14.0	0	1024	0
15	15	0.008857	1.0	15.0	0	1024	0
16	16	0.008876	1.0	16.0	0	1024	0
17	17	0.009895	1.0	17.0	0	1024	0
18	18	0.009922	1.0	18.0	0	1024	0
19	19	0.009943	1.0	19.0	0	1024	0

	Iteration	Time	Fitness	FEvals	Restarts	max_iters	current_restart
0	0	0.000202	1.0	0.0	0	1024	0

Results for SA on Four Peaks Problem:

	Iteration	Time	Fitness	FEvals	Temperature	max_iters
0	0	0.000075	1.0	0.0	0.1	1024
1	1	0.001336	1.0	2.0	0.1	1024
2	2	0.002470	1.0	4.0	0.1	1024
3	3	0.003571	1.0	6.0	0.1	1024
4	4	0.003604	1.0	8.0	0.1	1024
5	5	0.004693	1.0	10.0	0.1	1024
6	6	0.004725	1.0	12.0	0.1	1024
7	7	0.004754	1.0	14.0	0.1	1024
8	8	0.004782	1.0	16.0	0.1	1024
9	9	0.005887	1.0	18.0	0.1	1024
10	10	0.005920	1.0	20.0	0.1	1024
11	11	0.005949	1.0	22.0	0.1	1024
12	12	0.005978	1.0	24.0	0.1	1024
13	13	0.006007	1.0	26.0	0.1	1024
14	14	0.006036	1.0	28.0	0.1	1024
15	15	0.006064	1.0	30.0	0.1	1024
16	16	0.006093	1.0	32.0	0.1	1024
17	17	0.007242	1.0	34.0	0.1	1024
18	18	0.007277	1.0	36.0	0.1	1024
1025	0	0.000075	1.0	0.0	0.5	1024

	Iteration	Time	Fitness	FEvals	Temperature	max_iters
0	0	0.000075	1.0	0.0	0.1	1024
1025	0	0.000075	1.0	0.0	0.5	1024
2050	0	0.000075	1.0	0.0	0.75	1024
3075	0	0.000075	1.0	0.0	1.0	1024
4100	0	0.000075	1.0	0.0	2.0	1024
5125	0	0.000075	1.0	0.0	5.0	1024

Results for GA on Four Peaks Problem:

	Iteration	Time	Fitness	FEvals	Population Size	Mutation Rate	max_iters
0	0	0.002059	1.0	200.0	200	0.1	1024

This reached peak optimal state in one iteration hence didnt want to do any more optimization. Hence this is the best state as well.

Results for MIMIC on Four Peaks Problem:

	Iteration	Time	Fitness	FEvals	use_fast_mimic	Population Size	Keep Percent	max_iters
0	0	0.001978	1.0	200.0	False	200	0.2	1024

Same analysis as the above one.

It seems like in this Problem MIMIC and GA found the optimal level pretty quick, but the others were trying to optimize the results more, which tells us that there is a possibility that more iterations could have helped the other algorithms.

Results for RHC on Knapsack Problem:

	Iteration	Time	Fitness	FEvals	Restarts	max_iters	current_restart
0	0	0.000096	21.016413	0.0	0	1024	0
1	1	0.000861	21.016413	1.0	0	1024	0

	Iteration	Time	Fitness	FEvals	Restarts	max_iters	current_restart
0	0	0.000096	21.016413	0.0	0	1024	0

Reached optimal levels soon.

Results for SA on Knapsack Problem:

	Iteration	Time	Fitness	FEvals	Temperature	max_iters
4365	5	0.003376	20.376447	10.0	5.0	1024

Results for GA on Knapsack Problem:

	Iteration	Time	Fitness	FEvals	Population Size	Mutation Rate	max_iters
0	0	0.003136	21.016413	200.0	200	0.1	1024

Results for MIMIC on Knapsack Problem:

	Iteration	Time	Fitness	FEvals	use_fast_mimic	Population Size	Keep Percent	max_iters
0	0	0.003025	21.016413	200.0	False	200	0.2	1024

It seems like all of them reached optimal levels very soon, but SA and RHC didnt really get to their best most probably due to the fac they got caught at a local maxima, but they did better than last time.

Results for RHC on Traveling Salesman Problem:

	Iteration	Time	Fitness	FEvals	Restarts	max_iters	current_restart
1008	1008	1.397031	24.99668	1128.0	0	1024	0
1009	1009	1.398370	24.99668	1129.0	0	1024	0
1010	1010	1.399703	24.99668	1130.0	0	1024	0
1011	1011	1.401040	24.99668	1131.0	0	1024	0
1012	1012	1.402382	24.99668	1132.0	0	1024	0
1013	1013	1.403712	24.99668	1133.0	0	1024	0
1014	1014	1.405050	24.99668	1134.0	0	1024	0
1015	1015	1.406396	24.99668	1135.0	0	1024	0
1016	1016	1.407728	24.99668	1136.0	0	1024	0
1017	1017	1.409064	24.99668	1137.0	0	1024	0
1018	1018	1.410405	24.99668	1138.0	0	1024	0
1019	1019	1.411759	24.99668	1139.0	0	1024	0
1020	1020	1.413102	24.99668	1140.0	0	1024	0
1021	1021	1.414440	24.99668	1141.0	0	1024	0
1022	1022	1.415776	24.99668	1142.0	0	1024	0
1023	1023	1.417112	24.99668	1143.0	0	1024	0
1024	1024	1.418445	24.99668	1144.0	0	1024	0

	Iteration	Time	Fitness	FEvals	Restarts	max_iters	current_restart
1008	1008	1.397031	24.99668	1128.0	0	1024	0

Results for SA on Traveling Salesman Problem:

	Iteration	Time	Fitness	FEvals	Temperature	max_iters
1939	914	1.255504	24.035166	1066.0	0.5	1024
1940	915	1.256849	24.035166	1067.0	0.5	1024
1941	916	1.259317	24.035166	1068.0	0.5	1024
1942	917	1.260671	24.035166	1069.0	0.5	1024
1943	918	1.262011	24.035166	1070.0	0.5	1024
1944	919	1.263361	24.035166	1071.0	0.5	1024
1945	920	1.264723	24.035166	1072.0	0.5	1024
1946	921	1.266068	24.035166	1073.0	0.5	1024
1947	922	1.267411	24.035166	1074.0	0.5	1024
1948	923	1.268751	24.035166	1075.0	0.5	1024
1949	924	1.270097	24.035166	1076.0	0.5	1024
1950	925	1.271479	24.035166	1077.0	0.5	1024
1951	926	1.272835	24.035166	1078.0	0.5	1024
1952	927	1.274180	24.035166	1079.0	0.5	1024
1953	928	1.276615	24.035166	1080.0	0.5	1024
1954	929	1.277957	24.035166	1081.0	0.5	1024
1955	930	1.279301	24.035166	1082.0	0.5	1024
1956	931	1.281735	24.035166	1083.0	0.5	1024
1957	932	1.283106	24.035166	1084.0	0.5	1024
1958	933	1.284499	24.035166	1085.0	0.5	1024

	Iteration	Time	Fitness	FEvals	Temperature	max_iters
1939	914	1.255504	24.035166	1066.0	0.5	1024

Results for GA on Traveling Salesman Problem:

	Iteration	Time	Fitness	FEvals	Population Size	Mutation Rate	max_iters
1020	1020	268.800930	12.865378	205449.0	200	0.1	1024
1021	1021	269.062779	12.865378	205650.0	200	0.1	1024
1022	1022	269.319429	12.865378	205851.0	200	0.1	1024
1023	1023	269.579529	12.865378	206052.0	200	0.1	1024
1024	1024	269.865889	12.865378	206253.0	200	0.1	1024

	Iteration	Time	Fitness	FEvals	Population Size	Mutation Rate	max_iters
1020	1020	268.80093	12.865378	205449.0	200	0.1	1024

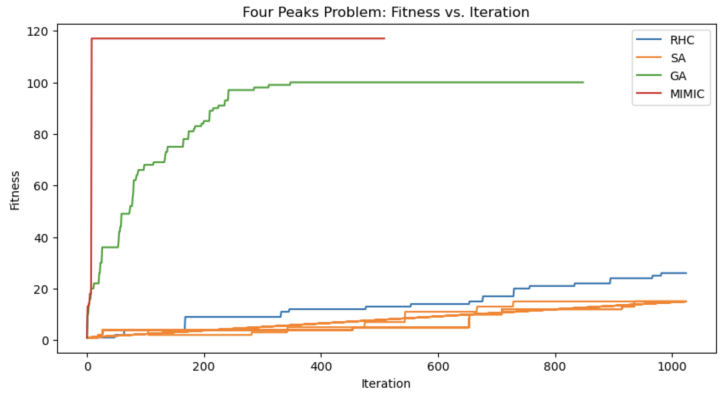
Results for MIMIC on Traveling Salesman Problem:

	Iteration	Time	Fitness	FEvals	use_fast_mimic	Population Size	Keep Percent	max_iters
1	1	4.371920	44.735995	402.0	False	200	0.2	1024
2	2	8.178668	44.735995	603.0	False	200	0.2	1024
3	3	11.867631	44.735995	804.0	False	200	0.2	1024
4	4	15.390757	44.735995	1005.0	False	200	0.2	1024
5	5	19.057375	44.735995	1206.0	False	200	0.2	1024
6	6	22.627864	44.735995	1407.0	False	200	0.2	1024
7	7	26.158754	44.735995	1608.0	False	200	0.2	1024
8	8	29.737841	44.735995	1809.0	False	200	0.2	1024
9	9	33.256110	44.735995	2010.0	False	200	0.2	1024
10	10	36.781922	44.735995	2211.0	False	200	0.2	1024
11	11	40.331950	44.735995	2412.0	False	200	0.2	1024
12	12	43.883096	44.735995	2613.0	False	200	0.2	1024
13	13	47.416925	44.735995	2814.0	False	200	0.2	1024
14	14	50.967596	44.735995	3015.0	False	200	0.2	1024
15	15	54.494008	44.735995	3216.0	False	200	0.2	1024
16	16	58.032603	44.735995	3417.0	False	200	0.2	1024
17	17	61.553793	44.735995	3618.0	False	200	0.2	1024
18	18	65.126645	44.735995	3819.0	False	200	0.2	1024
19	19	68.651686	44.735995	4020.0	False	200	0.2	1024
20	20	72.178682	44.735995	4221.0	False	200	0.2	1024

	Iteration	Time	Fitness	FEvals	use_fast_mimic	Population Size	Keep Percent	max_iters
1	1	4.37192	44.735995	402.0	False	200	0.2	1024

We can see that all of them have tried to reach their optimal level, but it was GA that could reach it the quickest, MIMIC tried a lot to optimize the parameters but the the fitness level remained quite constant.

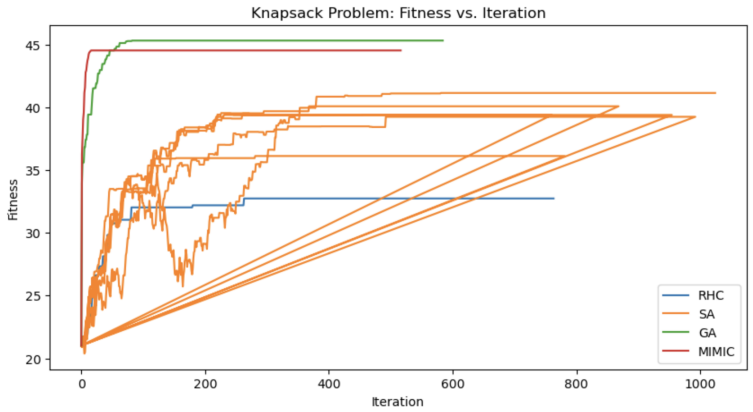
For each problem, I've run four different optimization algorithms and recorded their performance over multiple iterations. The Fitness value indicates how well the algorithm solved the problem at each iteration.



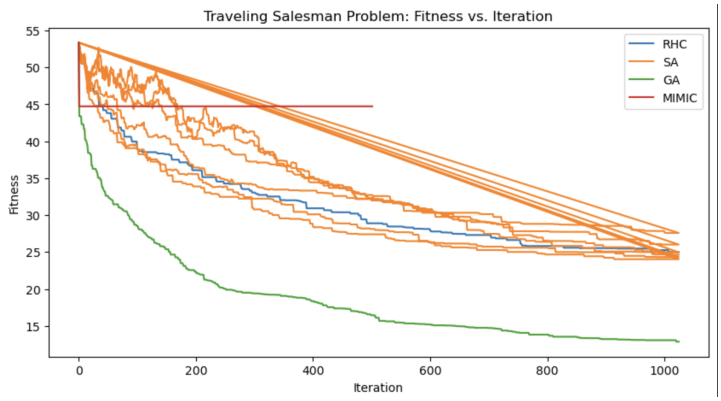
MIMIC : achieved the highest fitness score. This suggests that MIMIC was most successful at finding an optimal solution for the Four Peaks Problem in this particular run because it didn't get stuck at a local minima due to its probabilistic modeling.

GA : achieved the second-highest fitness score. This indicates that GA was also able to find a good solution, although it didn't perform as well as MIMIC but it didn't get stuck at a local minima due to its crossover and mutation .

RHC : achieved the third-highest fitness score, but it didnt a reasonably good solution and was outperformed by both MIMIC and GA, it probably got stuck at a local minima.
SA :achieved the lowest fitness score. This suggests that SA was the least successful at finding an optimal solution for this problem compared to the other algorithms.



In this graph we can see that GA did the best closely followed by MIMIC, then SA (which did much better than the 4 peaks problem, which i believe is attributed to its property of being allowed to move down time to time from your local maxima) and RHC did the worst.
I expanded the SA to understand how everything did the jagged lines represent the first few iterations and params, and the straight line represents the best/optimal params.



GA performed the best among all the algorithms. This could be due to GA's ability to maintain a diverse population of solutions and generate new ones through crossover and mutation, which is particularly useful in a problem like the Traveling Salesman where the solution space is large and complex. SA and RHC did almost identical, and much better than MIMIC. MIMIC did the worst because of the complicated dependencies in the travelling salesman problem.

MIMIC excelled in the Four Peaks problem due to its effective probabilistic modeling. GA demonstrated robust performance across all problems, with its mechanisms of crossover and mutation proving particularly effective. SA showed notable improvement in the Knapsack problem, likely due to its ability to escape local optima. RHC found reasonably good solutions but appeared to be trapped in local optima in some cases.

In conclusion, each algorithm has its strengths and weaknesses, and their performance can vary greatly depending on the specific characteristics of the problem at hand.
Improvements could be utilizing advanced hyperparameter techniques, also we could have used hybrid techniques like combining 2 algorithms something along the lines of utilizing GA for initial exploration and then using SA for exploitation and refinement.

PART 2

For the second part of the report. I used a new dataset for this as I felt it would be a good practice for me to work with something new. I chose the Breast Cancer Wisconsin Dataset, which has 569 instances, and its almost noise free and has minimal empty values. I also thought this dataset had a lot of real world implications so I picked it.

I built a model with 3 layers, Input hidden and an output layer. I utilized hyperparameter tuning by using Keras Hyperband and got the most optimal hyperparameters as well as the best learning rate and the number of epochs. I

retrained the model with the best stats and got the accuracy to be 90.35%

```
[test loss, test accuracy]: [0.31278082728385925, 0.9035087823867798]
```

Utilizing keras was really nice as it only took 5 seconds for tuning and training.

Next step I trained the same architecture of the model using the 3 algorithms (RHC, SA, GA). Since I utilized hyperparam tuning for the traditional method, it only seemed right that i would use it here too. The hyperparameters tuned for RHC include the number of restarts. Restarts in RHC allow the algorithm to jump to a new random position in the search space when it gets stuck in a local optimum. The grid search parameters for RHC included restarts with values [0, 5, 10]. This means the algorithm was tested with 0, 5, and 10 restarts to see which one yields the best performance.

The hyperparameters tuned for SA include the decay schedule which determines how the temperature decreases over time. A slower decay can allow more exploration of the solution space but may also increase the chance of getting stuck in a local optimum.

The hyperparameters tuned for GA include population size and mutation probability. The population size determines the number of solutions that are explored in each generation, and the mutation probability determines how often a solution is randomly changed.

In all three algorithms, other hyperparameters related to the neural network itself were also tuned. These include the learning rate ,activation function, and max iterations.

After Hyperparameter Tuning we got the results:

```
Test accuracy with best RHC hyperparameters: 0.6228070175438597
Test accuracy with best SA hyperparameters: 0.6228070175438597
Test accuracy with best GA hyperparameters: 0.9649122807017544
```

It took around 9 minutes for all 3 hyperparameter tuning and fitting.

In backtracking, the traditional method, it uses gradient descent to minimize error. And the error is propagated to the previous layers (hence the name backpropagation).
In RHC, we improve the accuracy by using iterative techniques. It checks its neighborhood for better solutions, if a better solution is found, it moves to that solution and repeats the process until no better solution is found.

In SA, this is a probabilistic technique along with iterativeness used for finding an approximate global optimum. It uses a random search instead of a gradient descent to avoid being trapped in local minima.

GA uses operations such as crossover, mutation, and selection to generate solutions to optimization problems and reach the optima.

Accuracy :

GA > Traditional > RHC = SA

The reason for this might be the ability of GA to explore a larger solution space and avoid local minima more effectively than the other methods.

On the other hand, both RHC and SA achieved significantly lower accuracies compared to backpropagation and GA. This could be because these methods can easily get stuck in local optima, especially in complex high-dimensional spaces such as those encountered in neural networks.

The reason GA might have performed better than back prop might also be due to the fact, GA performs Global Search instead of just the local search that is performed by Backprop.
It also uses population diversity, even if some get stuck at a local minima others might not and will keep growing, the techniques used by GA also makes it less prone to overfitting.

Hyperparameter tuning played a crucial role in the performance of all algorithms. It allowed us to optimize the parameters of each algorithm to achieve the best possible performance on the given task. However, it also increased computational cost as more combinations needed to be tested.

Overall, this study highlights the importance of choosing the right optimization algorithm and tuning hyperparameters effectively when training neural networks. While backpropagation is a good default choice, alternative methods like GA can sometimes yield better results, especially for complex tasks or when global optima are needed. However, these methods also come with their own trade-offs in terms of computational cost and complexity.

Future work could explore other optimization algorithms or variations of the current ones, use more sophisticated techniques for hyperparameter tuning, or apply these methods to different types of neural networks and tasks. This would provide further insights into the strengths and weaknesses of each method and help us develop more effective strategies for neural network training.