Optimization

Mahmood Tabesh, Mtabesh6@gatech.edu

Introduction:

In this homework we practice exploring optimization algorithms. Optimization algorithms can be used to solve a wide range of problems in machine learning, including parameter estimation, feature selection, model selection, and hyperparameter tuning. However, the choice of optimization algorithm depends on the nature of the problem being solved and the constraints and objectives involved.

Optimization models:

Four common randomized optimization models will be tested in this homework. In the following each model will be explained in brief.

Randomized Hill Climbing (RHC): is a simple optimization algorithm that starts with a random solution and iteratively makes small random perturbations to it to find an improved solution. It is suitable for problems with a single peak or a small number of peaks in the solution space.

Simulated Annealing (SA): is an optimization algorithm that is inspired by the process of annealing in metallurgy. It starts with a high-temperature state that allows for a large exploration of the solution space, and gradually decreases the temperature to converge towards the global optimum. It is suitable for problems with multiple peaks in the solution space.

Genetic Algorithm(GA): is a population-based optimization algorithm that is inspired by the process of natural selection. It involves generating a population of candidate solutions, selecting the fittest solutions for reproduction, and applying genetic operators such as mutation and crossover to create new solutions. It is suitable for problems with a large search space and multiple solutions.

MIMIC: (Mutual Information Maximization for Input Clustering) is a probabilistic optimization algorithm that uses a probability distribution over the solution space to generate new solutions. It starts with a random population of solutions and iteratively learns a probability distribution that models the structure of the solution space. It is suitable for problems with a complex search space and many variables.

All of these optimization algorithms are used to find the optimal solution to a problem, and each has its strengths and weaknesses depending on the nature of the problem being solved.

Problem 1 – One MAX:

The One Max problem is a simple and well-studied problem in the field of optimization and genetic algorithms. It is often used as a benchmark problem for evaluating the performance of optimization algorithms, particularly for binary optimization problems. The key challenge in solving the One Max problem is to balance the exploration and exploitation of the search space, to ensure that the algorithm can find the global maximum while avoiding getting stuck in local maxima.A one max problem of length 500 has been created with mlrose_hiive library in python. All the algorithms has been tested and hyper parameters has been tested to get the max fitness of each model. The maximum fitness for this problem is 500.

1.1. MIMIC:

As shown in Figure 1 the population size and keep percent were tested with different iteration levels. The population size in MIMIC determines the number of candidate solutions that are generated in each iteration. A larger population size generally leads to a better exploration of the solution space, as there are more candidate solutions to consider. However, a larger population size also increases the computational cost of the algorithm, as there are more solutions to evaluate and more memory required to store the population.

The keep percent parameter in MIMIC determines the fraction of the population that is retained from one iteration to the next. The remaining fraction of the population is replaced with new solutions generated from the learned probability distribution. A larger keep percent means that a larger fraction of the population is retained, and fewer new solutions are generated. This can lead to faster convergence to a good solution, but may also lead to premature convergence and a suboptimal solution.

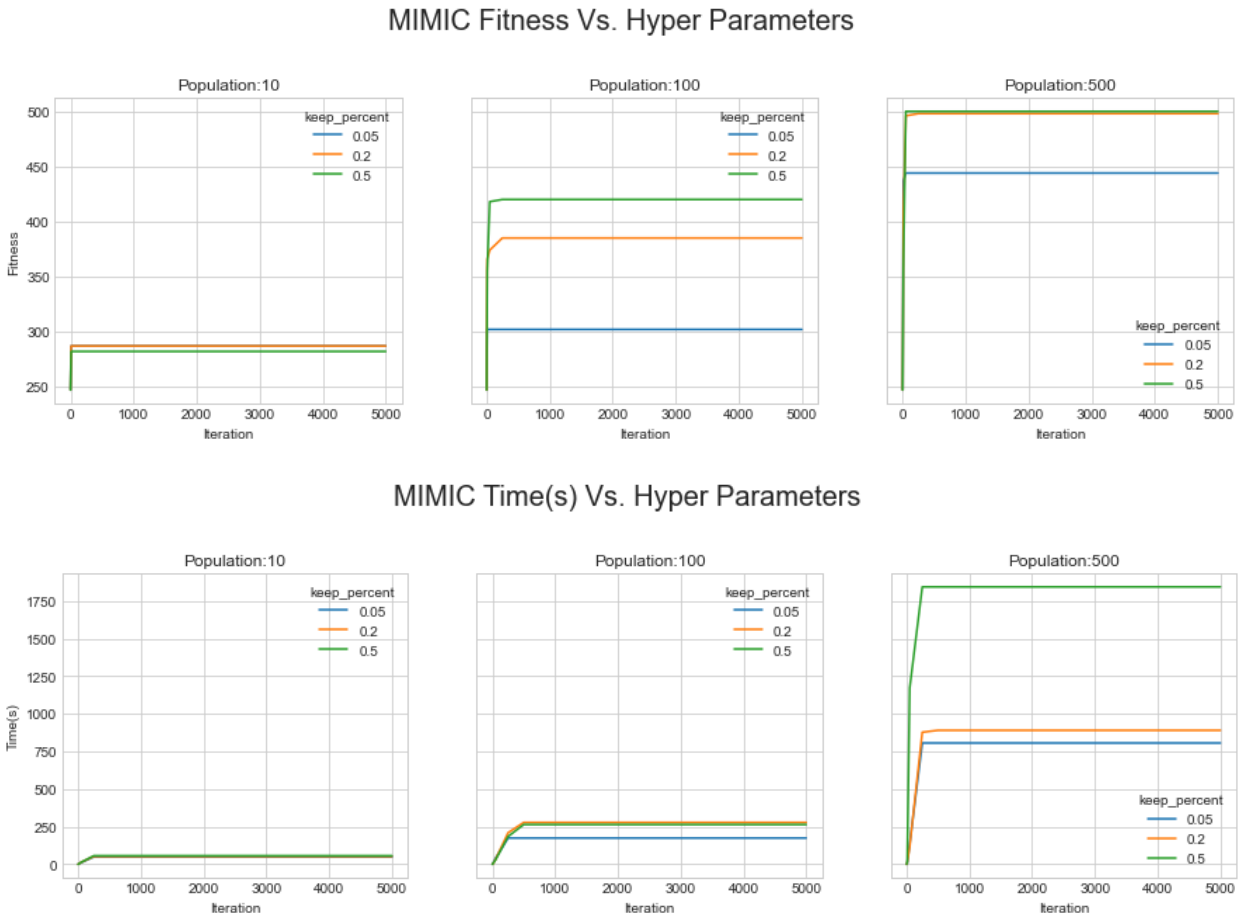


Figure 1 - P1 : MIMIC - Fitness and Time

1.2. Simulated Annealing (SA):

As shown in Figure 2Figure 1 the temperature and temperature decay type were tested with different iteration levels.

In SA, the temperature is a measure of the "energy" of the system and controls the probability of accepting a new solution. A higher temperature corresponds to a higher energy, which makes it more likely that the algorithm will accept a worse solution in the hope of finding a better one. As the algorithm progresses, the temperature is gradually reduced according to a cooling schedule, which reduces the probability of accepting worse solutions and encourages the algorithm to converge to a better solution.

In SA, the decay type determines how the temperature is reduced over time. The choice of decay type in SA affects the rate at which the temperature is reduced over time and can have a significant impact on the performance of the algorithm. Exponential decay typically leads to a faster decrease in temperature, which can be beneficial for problems that require a quick convergence. Arithmetic decay leads to a more gradual decrease in temperature, which can be beneficial for problems that require a more thorough exploration of the search space. Geometric decay can provide a good balance between the two approaches.

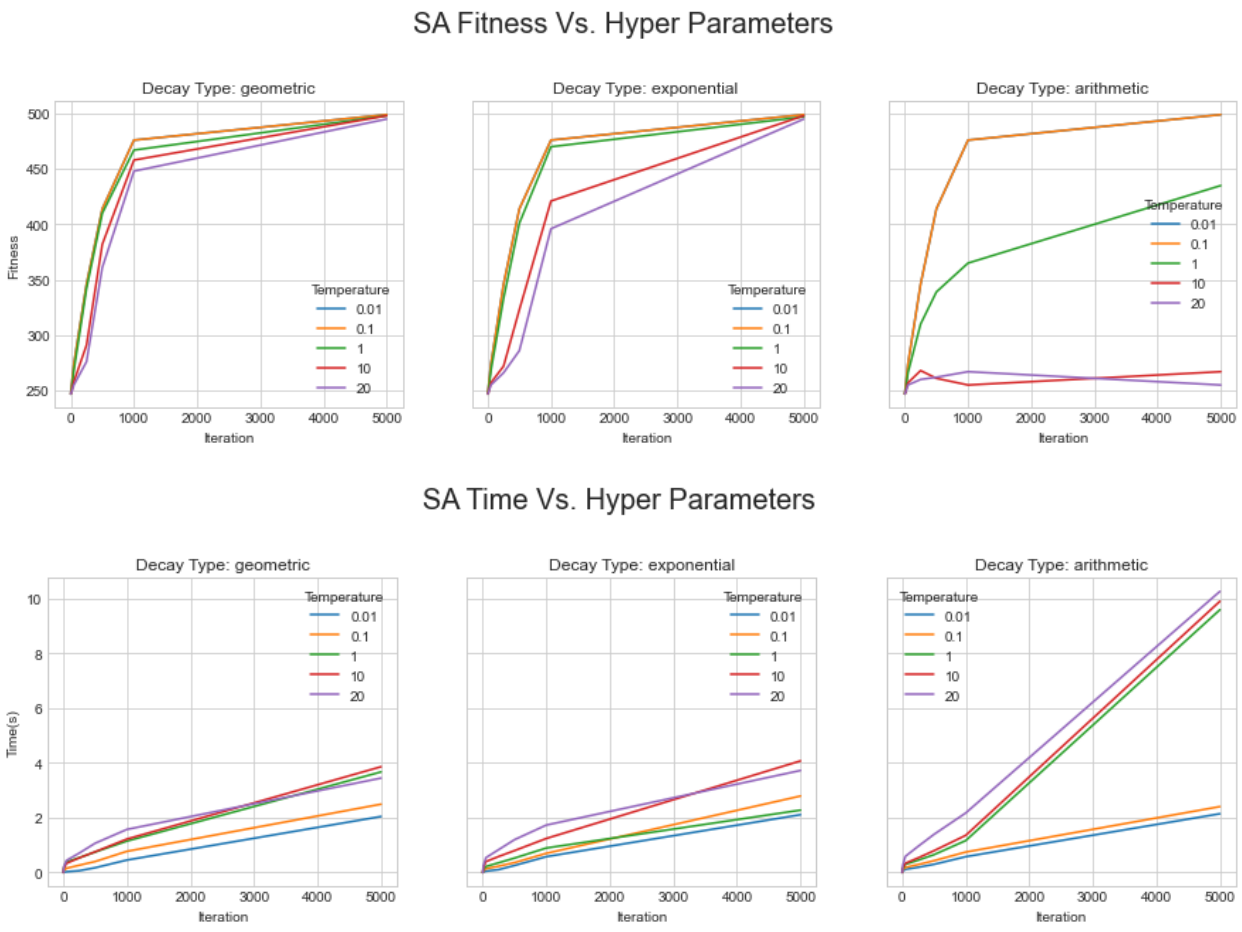


Figure 2- P1:SA - Fitness and Time

1.3. Randomized Hill climbing (RHC):

As shown in Figure 3 the number of random restart was tested with different iteration levels.

The number of random restarts is an important parameter in RHC. It determines how many times the algorithm should restart from a random initial solution if it gets stuck in a local optimum. When the algorithm reaches a local optimum, it is possible that further perturbations will not lead to an improvement in the objective function. In this case, the algorithm can be restarted from a new random initial solution to explore a different region of the search space. Increasing the number of random restarts in RHC can improve the quality of the solutions obtained by the algorithm. This is because more restarts increase the probability of finding the global optimum by exploring a larger portion of the search space. However, increasing the number of random restarts also increases the computational cost of the algorithm. Therefore, the number of random restarts should be chosen carefully to balance the trade-off between solution quality and computational cost.

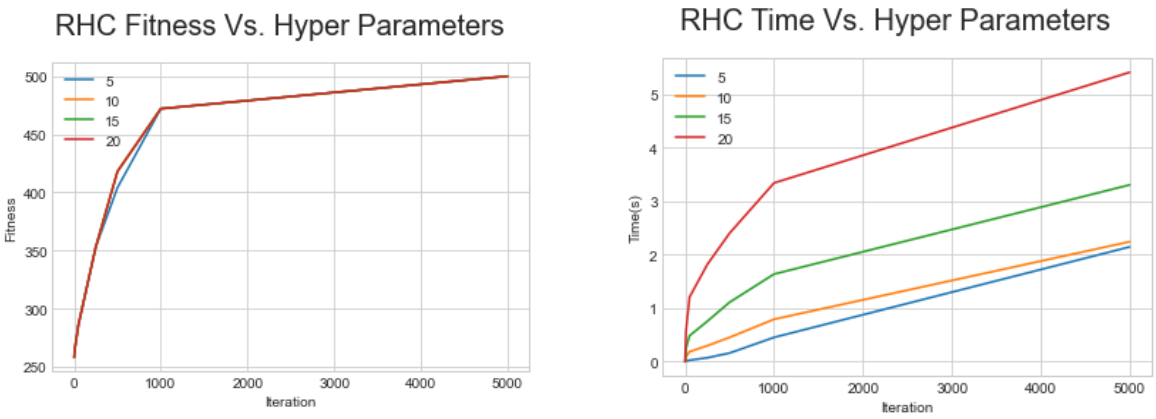


Figure 3 - P1: RHC- Fitness and Time

1.4. Genetic Algorithm (GA):

As shown in Figure 4 the population size and mutation rate was tested with different iteration levels. The population size has almost same impact as population size in MIMIC. The mutation rate can have a significant impact on the performance of a genetic algorithm. If the mutation rate is too low, there is a risk of premature convergence, where the population gets stuck in a local optimum and fails to explore the search space adequately. On the other hand, if the mutation rate is too high, the genetic algorithm can become too exploratory and may not converge to a good solution. A higher mutation rate can be useful in helping the algorithm avoid local optima and explore new regions of the search space. However, too high a mutation rate can lead to too much diversity in the population, making it difficult for the algorithm to converge to a solution. In general, the optimal mutation rate depends on the specific problem being solved, and it is often determined through empirical experimentation.

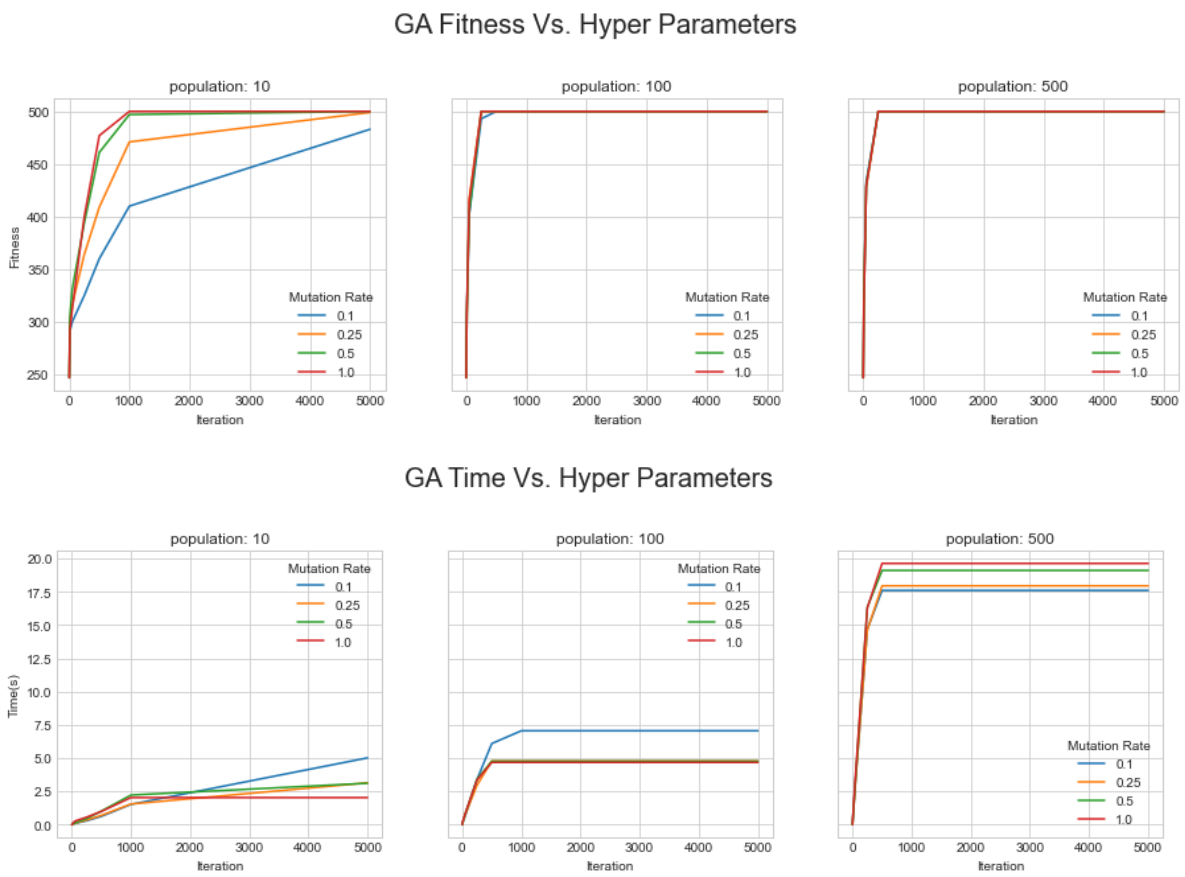


Figure 4 - P1:GA-Fitness Vs Time

1.5. Comparison of all Models:

In problem one, there is not any local optimum. There is just one global optimum. It is easy to solve. MIMIC and GA with 500 population will reach to maximum fitness in early iterations, however the computation time in MIMIC is much higher than other models. GA computation is lower than MIMIC. SA needs much more iteration; however it is much faster than other models.

The best candidate for problem 1, is Simulated Annealing following by genetic algorithms.

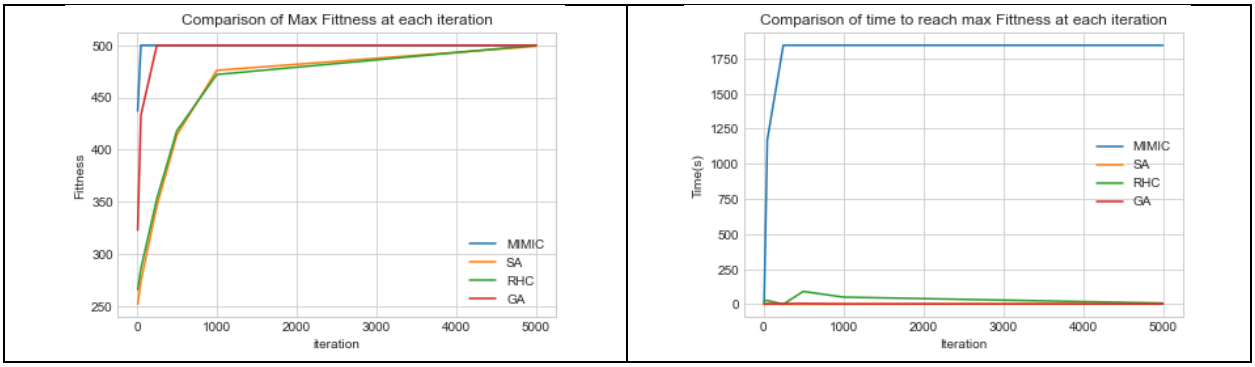


Figure 5 - P1 Comparison of Models

Problem2 – Four Peaks:

"Four Peaks" is a type of optimization problem that involves finding the maximum value of a function that has two global optima separated by a "rugged" fitness landscape. In this problem, there are two types of solutions: "short" solutions that have a low fitness but are close to one of the global optima, and "long" solutions that have a high fitness but are far from both global optima. The "Four Peaks" problem is so-called because the fitness landscape looks like four peaks and a valley between them. It is a challenging problem for optimization algorithms because it requires balancing exploration and exploitation to find both the short and long solutions. Many metaheuristic algorithms, such as genetic algorithms, simulated annealing, and hill climbing, have been applied to this problem to evaluate their performance in solving it. The "Four Peaks" problem is often used as a benchmark problem in optimization research to evaluate the effectiveness of different optimization algorithms. The problem is easy to understand and implement, yet it exhibits many of the characteristics of more complex real-world optimization problems. By studying the performance of algorithms on the "Four Peaks" problem, researchers can gain insights into how to design more effective optimization algorithms for real-world applications.

The defined problem is FourPack with length of 500. The parameter analysis were done in the code. But the results were not presented here due to limitation in pages. However, the final model performances were compared.

Model parameters of the best model are as follows:

- MIMIC Best parameters: {Population: 500, Keep Percent:0.2, iters:250, Fitness:39, Time: 141}
- SA Best parameters: {Temperature: 10, decay type: exponential, iters:5000, Fitness:24, Time: 11}
- RHC Best parameters: {number of restart: 20, iters:250, Fitness:11, Time: 0.37}
- GA Best parameters: {Population: 100, mutation:0.5, iters:1000, Fitness:500, Time: 61}

Model Comparison:

One of the challenges of the Four Peaks problem is that it has many local optima, which can trap optimization algorithms that rely on hill climbing or gradient-based methods. By maintaining a diverse population of candidate solutions and applying selection, crossover, and mutation operations, GA can explore different regions of the search space and escape from local optima. GA is able to handle rugged fitness landscapes like the Four Peaks problem, where the fitness function has many local optima and high levels of noise or uncertainty. GA is also able to handle problems with constraints, such as the requirement that the binary string has a certain number of 0s and 1s in the Four Peaks problem, by incorporating penalty functions or other constraint-handling mechanisms into the fitness evaluation process.

Also, the computation cost is higher than RHC and SA, however the maximum fitness that can be reached by GA much more than other models. The MIMIC model computation is still much higher than GA in this problem.

Finally, GA is a powerful optimization algorithm that is well-suited for solving Four Peaks problem that have complex, non-linear fitness functions and multiple local optima.

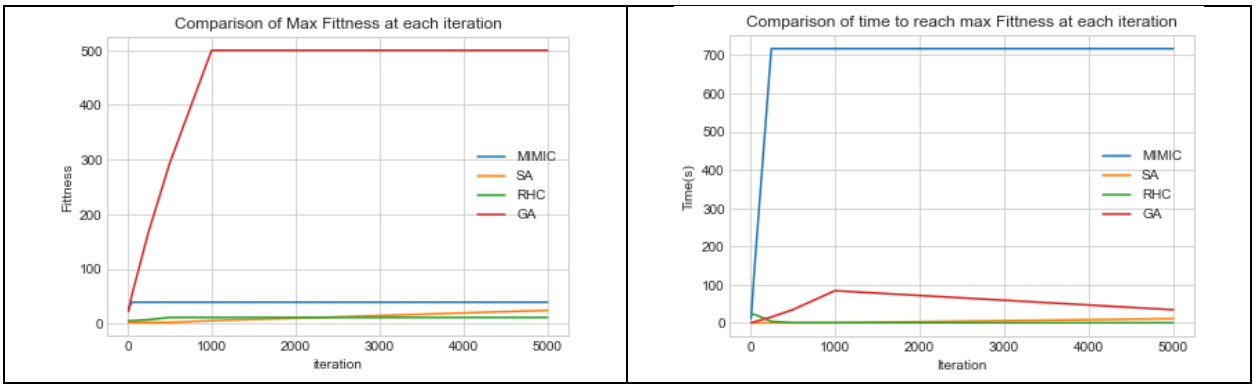


Figure 6 - Comparison of fitness and time in FourPeak Problem

Problem3 – Knapsack:

The Knapsack problem is a classic optimization problem in computer science and operations research. The problem is to find the most valuable combination of items that can fit into a knapsack with a given weight capacity. Each item has a weight and a value, and the objective is to maximize the total value of the selected items while ensuring that the total weight does not exceed the capacity of the knapsack. The Knapsack problem is a challenging problem because it is a combinatorial optimization problem, meaning that there are many possible combinations of items that could be selected. There are many variations of the Knapsack problem, including 0-1 Knapsack (where items can only be taken or left), fractional Knapsack (where items can be divided into fractions), and multiple Knapsack (where there are multiple knapsacks with different capacities). The Knapsack problem has many real-world applications, such as in logistics, financial planning, and resource allocation. For example, it could be used to optimize a delivery truck's cargo, by selecting the most valuable items that can fit within the weight capacity of the truck. Many optimization algorithms have been developed to solve the Knapsack problem, including dynamic programming, branch and bound, and heuristic/metaheuristic algorithms such as MIMIC and genetic algorithms.

The defined problem is Knapsack with length of 25. Weights range from 1 to 3 and values range from 1 to 10 . The parameter analysis were done in the code. But the results were not presented here due to limitation in pages. However, the final model performances were compared.

- MIMIC Best parameters: {Population: 500, Keep Percent:0.2, iters:10, Fitness:94, Time: 0.16}
- SA Best parameters: {Temperature: 1, decay type: arithmetic, iters:500, Fitness:83, Time: 0.16}
- RHC Best parameters: {number of restart: 20, iters:10, Fitness:74, Time:8.6}
- GA Best parameters: {Population: 100, mutation:0.5, iters:250, Fitness:94, Time: 0.41}

MIMIC is particularly effective for solving optimization problems that have a strong dependence on the interactions between different variables or features, such as combinatorial optimization problems and certain types of function optimization problems.

In this problem

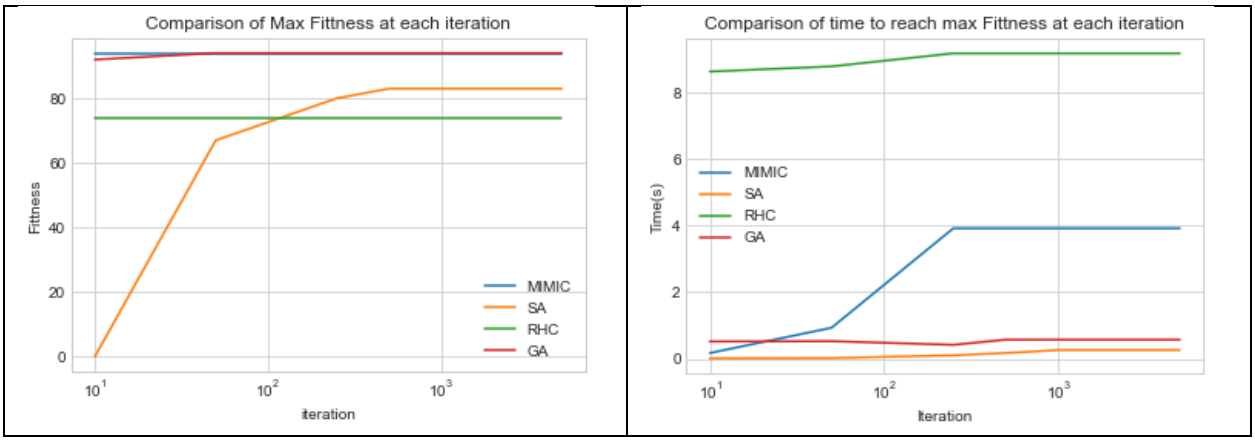


Figure 7 - Comparison of fitness and log of time in Knapsack Problem

MIMIC reached the highest fitness in very less iteration compared to GA. Finally, MIMIC is effective at solving optimization problems with complex fitness landscapes and strong interactions between the different variables or features.

Application of optimization models in neural network

All three optimization algorithms (SA, GA, RHC) can be applied to train neural networks. SA can be effective in avoiding local optima, GA can explore a large search space, and RHC is simple to implement but may get stuck in local optima. The performance of these algorithms depends on the specific problem and the tuning of their parameters.

Dataset:

In this dataset, information from almost 50000 US residents in 14 features is gathered. The ground truth is whether the income of the person is greater than $50k or less than that. Data initially extracted from the US Census Bureau [1]. The data sample were edited and uploaded in UCIU machine learning repo [2] for educational purposes.

Dataset has an unbalance data. For ease of use, under sampling has been done to make data balance. A series of data preprocessing has been done which can be found on the code. Finally, the train data has 800 sample points in training and 200 sample point in test set. Since target classes are balanced in dataset, accuracy were used as comparison metric for the evaluation.

NN with Randomized Hill Climbing:

RHC works by iteratively making small random changes to the weights and biases of the neural network and accepting the new solution if it improves the objective function. The changes are typically made by adding a small random value to the weight or bias. The learning rate and number of random restart point were tunned for this model. As shown below best model has learning rate of 0.1 and restart point of 20. It is much more sensitive to learning rate compare to number of restart point. Learning rate of greater than 0.1 does not that much effect on accuracy.

NN with Simulated annealing:

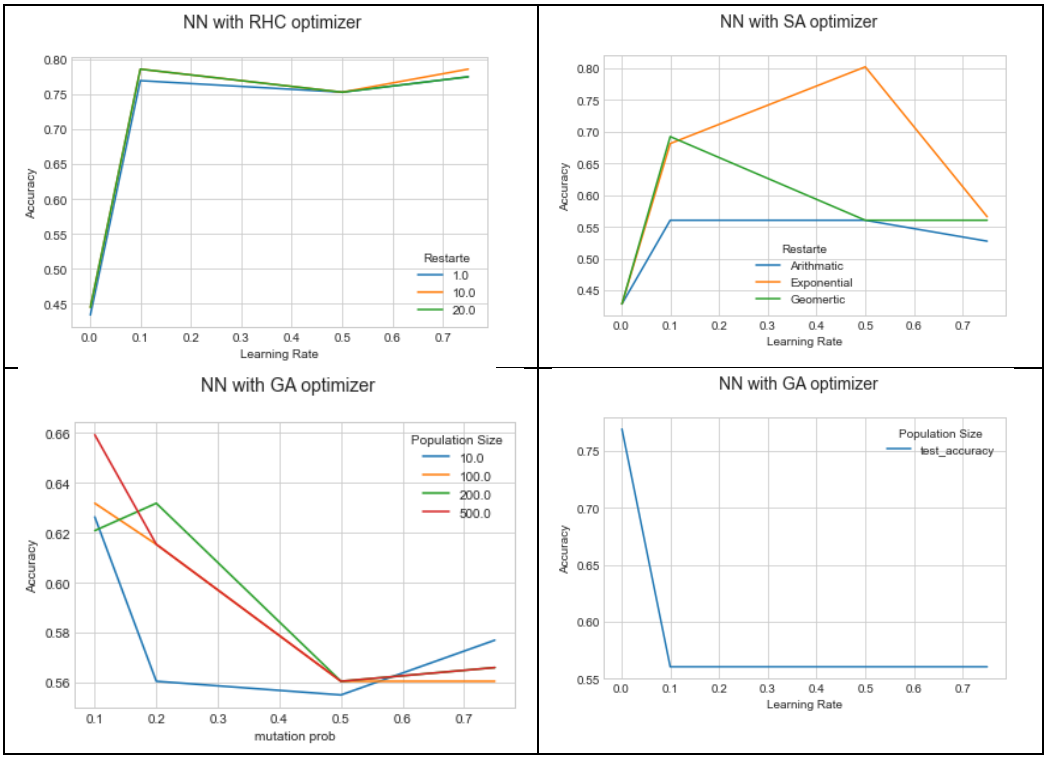
SA can avoid getting trapped in local optima and find better global optima compared to other optimization algorithms like Hill Climbing or Gradient Descent. The performance of SA depends on the temperature schedule and the initial temperature. Choosing the appropriate temperature schedule and initial temperature can be a challenging task. The learning rate and schedule type were tuned for this model. As shown below best model has learning rate of 0.5 and with exponential temperature decay.

NN with Genetic Algorithm:

Using a genetic algorithm as an optimizer for neural network training can have several effects. GA can help to explore a larger search space, which can be beneficial in finding a good set of weights for the neural network.GA can help to avoid getting stuck in local minima by allowing the algorithm to search for multiple solutions at the same time.GA can work well for problems with a large number of variables or where the search space is not well-defined. Mutation probability and population size are important factors in GA. These parameters were tuned. Population of 500 with mutation of 0.1 gives the highest accuracy. Accuracy will decrease by increasing the mutation probability.

NN with Gradient Descent:

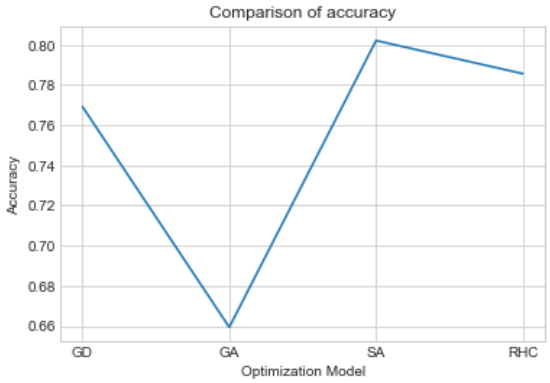
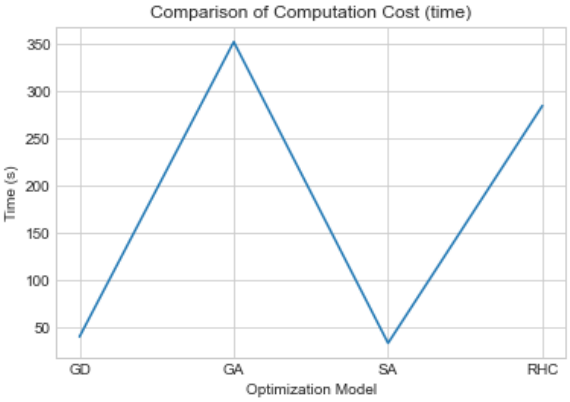
Gradient Descent (GD) is a commonly used optimization algorithm in neural networks. I didn't submit HW1. Thus to have a solid comparison of optimizers together, I was using gradient descent for optimization of weights in neural network. I tunned learning rate for this method of optimization. Learning rate of 0.001 gave the best accuracy.



Comparison of performances

SA has the highest accuracy. It can avoid getting trapped in local optima and find better global optima compared to other optimization algorithms like Hill Climbing or Gradient Descent

RHC and GD have almost the same accuracy, however, accuracy of GA is lower than other methods. Computation cost of GA is also high but computation cost of the SA and GD are much lower than other methods. SA can find the global optimum fast and with better performance.

Accuracy	**Computation Cost**
	

	Computation Cost	Accuracy in NN
Gradient Descent	Low	Medium
Genetic Algorithm	High	Low
Simulated Annealing	Low	High
Randomized Hill Climbing	High	Medium

In general, **Gradient Descent (GD)** is beneficial for neural networks when the cost function is differentiable and the dataset is large, as it efficiently finds the global minimum of the cost function.

Randomized Hill Climbing (RHC) is beneficial for neural networks when the cost function is discontinuous, non-differentiable, or noisy, as it can escape local optima and find a good solution.

Simulated Annealing (SA) is beneficial for neural networks when the cost function is noisy and contains many local optima, as it can explore the search space more extensively and can occasionally accept worse solutions to escape from local optima.

Genetic Algorithms (GA) is beneficial for neural networks when the cost function is non-differentiable or multi-modal, as they can explore the search space more widely and can handle constraints and binary variables more easily.