Project 2 Randomized Optimization

Name: Luping Yang

GATech ID: lyang38

***Abstract*— Optimization algorithms play an important role in artificial intelligence, especially randomized search algorithms. In the project, four randomized search algorithms are applied to three discrete problems and one artificial neural network. The four randomized search algorithms are randomized hill climbing (RHC), simulated annealing (SA), generic algorithm (GA), and Mutual Information Maximizing Input Clustering (MIMIC). Then the behavior, performance, and efficiency of these algorithms are compared to gain intuition of these algorithms.**

Keywords*— *randomized hill climbing (RHC), simulated annealing (SA), generic algorithm (GA), and Mutual Information Maximizing Input Clustering (MIMIC).

I. Introduction to Problems

A. *Four Peaks*

The four peak problem is taken from Charles L. Isbell. Given an N-dimensional vector X, the four peaks evaluation function is defined as:

$$f(\vec{X},T) = \max\left[tail(0,\vec{X}), head(1,\vec{X})\right] + R(\vec{X},T)$$

where

$$\begin{aligned} tail(b,\vec{X}) &= \text{number of trailing } b\text{'s in } \vec{X} \\ head(b,\vec{X}) &= \text{number of leading } b\text{'s in } \vec{X} \end{aligned}$$

$$R(\vec{X},T) = \begin{cases} N & \text{if } tail(0,\vec{X}) > T \text{ and } head(1,\vec{X}) > T \\ 0 & \text{otherwise} \end{cases}$$

For four peaks problem, there are two global maxima in the objective function. There are also sub-optimal local maxima that could potentially trap the optimization algorithms. As the values of T increases, the difficulty of this problem increases significantly because the regions of the local maxima are large with large T values. Thus, it is interesting to see how the four algorithms solve the four peaks problem.

For this problem, the threshold parameter for Four Peaks fitness function, expressed as a percentage of the state space dimension, is set to 0.1. The number of elements in state vector is set to 100.

B. *Flip-Flop Problem*

In Flip-Flop problem, the objective is to maximize the flipped bits in a bit string. Thus, the objective function is the sum of flipped bits, for example the number of flips in 1011 is 2 and the number of flips in 1000 is 1.

For the Flip-Flop problem, the number of bits is set to be 300 to make this problem hard enough for the four optimization algorithms.

C. *Knpsack Problem*

Accord to Wikipedia, the knapsack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

The complexity of Knapsack problem is NP-Complete. The Knapsack problem has dense local maximal values, which proves to be a big challenge for most searching algorithms.

For this problem, the weight and values are uniformly sample in the range of 10 to 50. Then the weight is normalized to be summed up to 1. The maximum weight is set to 0.5. The length of the problem is 100 to make this problem is hard to optimize.

II. Four Peaks (Best for GA)

In this section, I document the hyper-parameter search, fitness, and efficiency of the four algorithms for the Four Perks problem.

A. *Hyper-Parameter Search*

For Randomized Hill Climbing, I used the following hyper-parameters: iteration_list = [10000], restart_list = [100], max_attempts = 100. There is not much to tune for RHC.

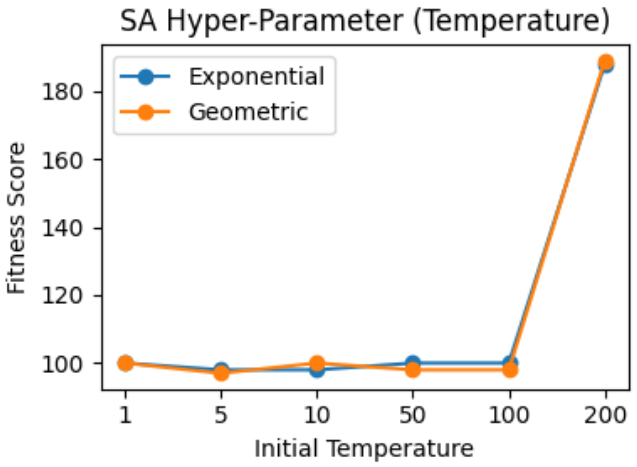


Fig. 1. Hyper-parameter tuning for SA in Four Peaks problem.

The Simulated Annealing (SA) hyper-parameters tuned are decay (ExpDecay and GeomDecay) and init_temperature ([1, 5, 10, 50, 100, 200]). From Figure 1, it is observed that the SA algorithm is not sensitive to the decay parameter, but very sensitive to init_temperature. It is reasonable that SA being

sensitive to init_temperature, since a good init_temperature parameter can help SA balance exploration and exploitation. It is better for SA to explore more in early search, which is equivalent to having a large temperature.

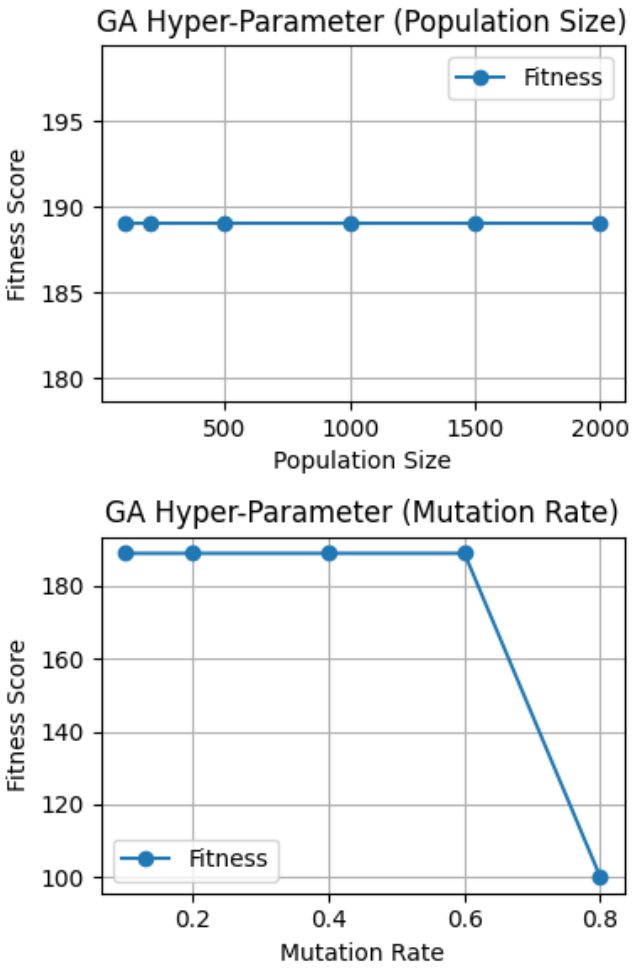


Fig. 2. Hyper-parameter tuning for GA in Four Peaks problem.

For Generic Algorithm (GA), the following parameters are tuned: population_sizes = [100, 200, 500, 1000, 1500, 2000], mutation_rates = [0.1, 0.2, 0.4, 0.6, 0.8].

The population size is the size of population to be used in genetic algorithm. From Figure 2 upper plot, it is observed that GA is not sensitive to population size in Four Peaks problem.

Mutation rate is the probability of a mutation at each element of the state vector during reproduction. The mutation rate allows GA to explore potential improvements. Thus, higher mutation rate is more likely for GA to get out of local optima and high chance to find global optima. But if the mutation rate is too high, it is hard for GA to stabilize, even when GA has already arrived global optima. The lower plot in Figure 2 confirms our understanding about mutation rate. The GA works for Four Peak problem when mutation rate in range (0.1, 0.6), but when mutation rate is too high, i.e. 0.8, the fitness score drops significantly.

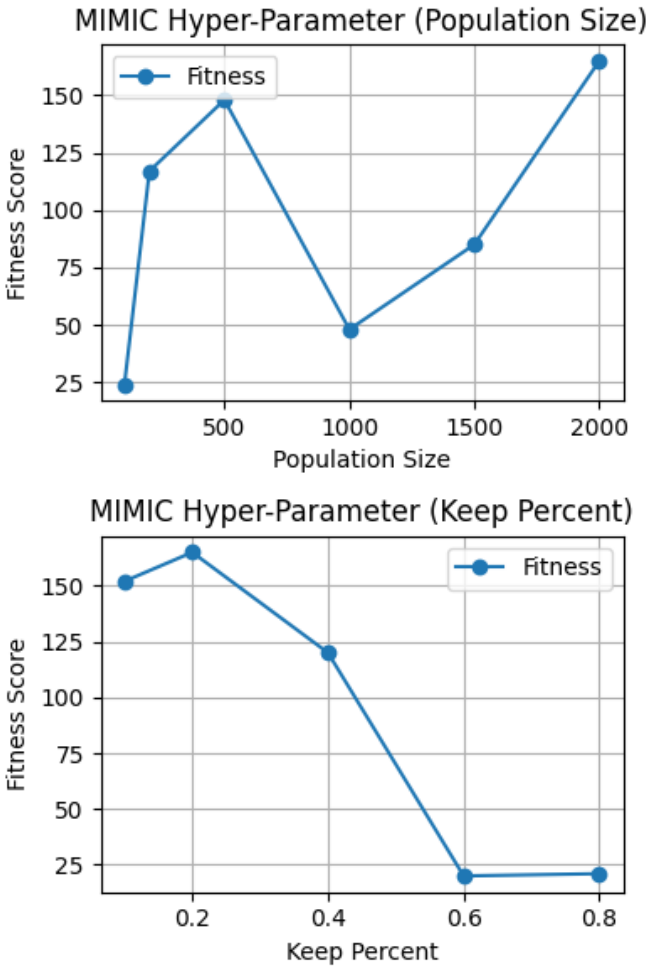


Fig. 3. Hyper-parameter tuning for MIMIC in Four Peaks problem.

For MIMIC, the following parameters are searched: population_sizes = [100, 200, 500, 1000, 1500, 2000], keep_percent_list=[0.1, 0.2, 0.4, 0.6, 0.8].

From Figure 3 upper plot, it is observed that the fitness scores of MIMIC generally increase with population size.

The keep percent parameter controls how that MIMIC explores and exploits. The larger the keep percent the less MIMIC explores and more likely to trap in local optima; whereas the smaller the keep percent the more MIMIC explores and more likely to get out of local optima. From Figure 3 lower plot, it is observed that the best keep percent for Four Peaks problem is 0.2 and the fitness score decrease significantly as keep percent increase to 0.8.

B. Fitness Score Comparison

In Figure 4, the fitness curves of the four algorithms with their best tunned parameters are plotted.

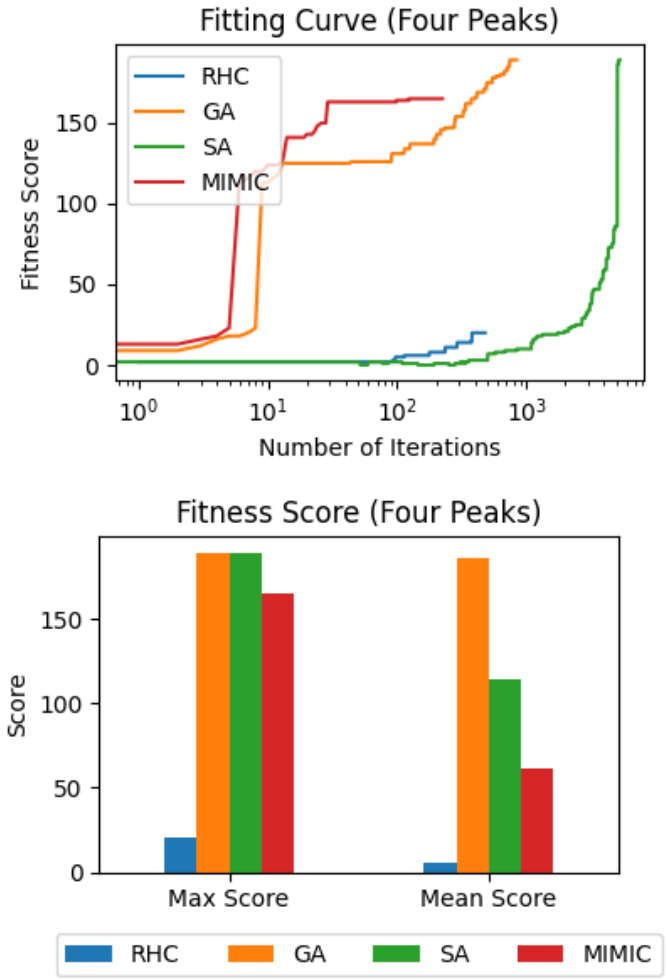


Fig. 4. Fitness score comparison for Four Peaks problem.

From Figure 4, it is observed that RHC did the worst. RHC not only converges slowest but also has the lowest fitness sore.

Looking the at number of iterations in Figure 4, MIMIC converges the most quickly of all four algorithms. But MIMIC converges to a local optimum, which is significantly smaller than global optima.

In Figure 4, the SA finds the global optima, but SA took much more iterations to find the global optima. One interesting observation about SA is that the fitness score of SA goes below the initial fitness score. This is reasonable to SA, since SA has certain probability to go to some locations, where have lower fitness score.

In Figure 4, it is noted that GA is the best algorithm for Four Peaks problem. GA not only can find the global optima but also can find it with relatively small number of iterations. This is because the crossover and mutation of GA enable it to explore and improve fitness score.

It is noticeable that the mean score of GA is the same as its max score, meaning that GA is not sensitive to the hyper-parameters and GA can find global optimum in all most all cases, which is why I think GA is best for this Four-Peak problem.

C. *Efficiency Comparison*

Next, we take a close look at the efficiency of the algorithms. More specifically, we will examine the number of iterations and time used in maximization.

In Figure 5, we observe large variation in iteration number and time for RHC, which makes sense, since RHC is random search. For some cases, RHC gets trapped in local optima then quit, thus RHC uses small number of iteration and time. For some cases, RHC is on the road to find the better optima, thus uses larger number of iteration and time.

SA uses the greatest number of iterations of all algorithms, due to that fact that SA explores and improves gradually. But each iteration is quick. Thus, total time consumed is decent.

MIMIC uses the smallest number of iterations of all four algorithms. But each iteration takes a long time. Thus, total time is the longest of the four algorithms.

The number of iterations of GA is decent and the time of each iteration is decent. Thus, considering the effectiveness, the number of iteration and time used by GA is the best of all four algorithms.

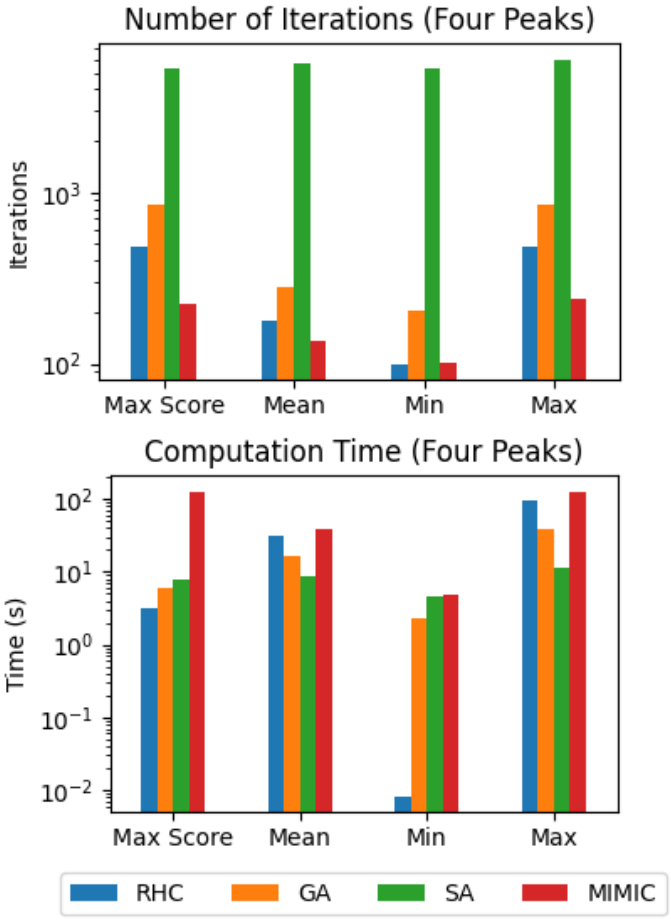


Fig. 5. Efficiency comparison for Four Peaks problem.

III. Flip-Flop (Best for SA)

In this section, I document the hyper-parameter search, fitness, and efficiency of the four algorithms for the Flip-Flop problem.

A. *Hyper-Parameter Search*

For Randomized Hill Climbing, I used the following hyper-parameters: iteration_list = [10000], restart_list = [100], max_attempts = 100.

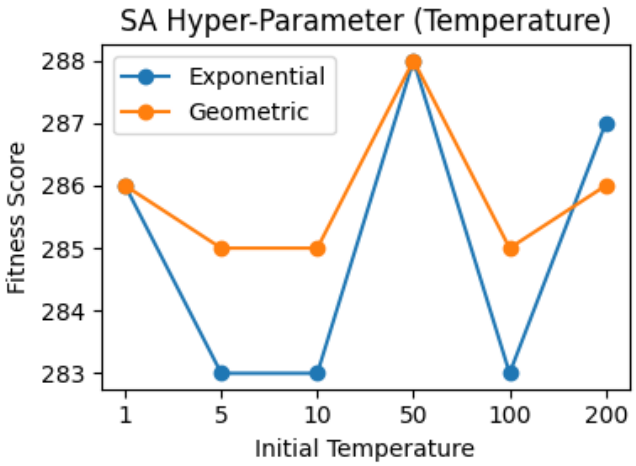


Fig. 6. Hyper-parameter tunning for SA in Flip-Flop problem.

The Simulated Annealing (SA) hyper-parameters tuned are decay (ExpDecay and GeomDecay) and init_temperature ([1, 5, 10, 50, 100, 200]). For Figure 6, it is observed that GeomDecay is better than ExpDecay for this problem.

The best initial temperature is 50 for this problem. For initial temperature being 50, SA can better balance exploration and exploitation.

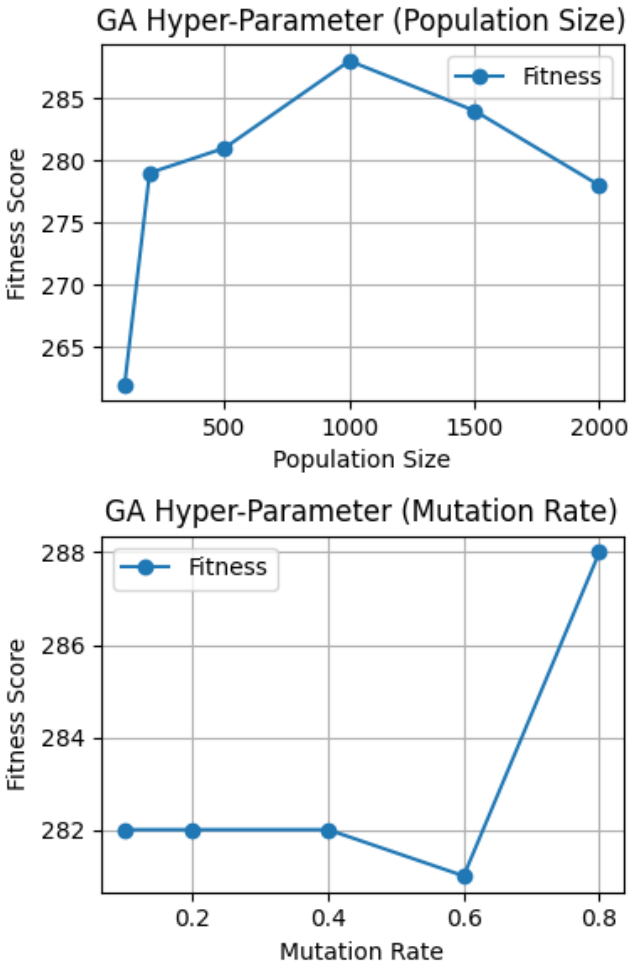
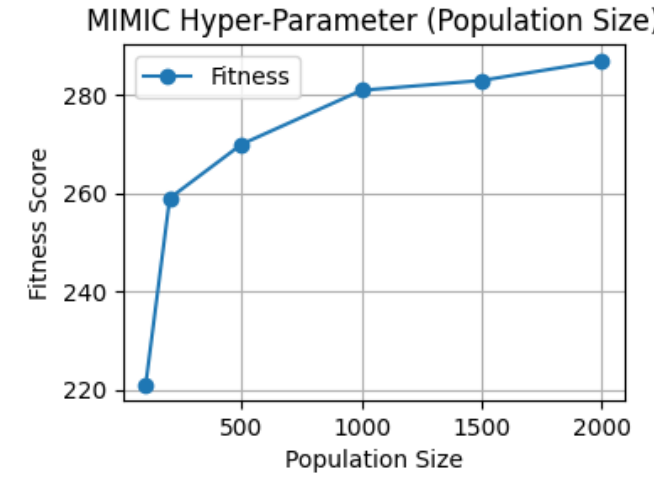


Fig. 7. Hyper-parameter tuning for GA in Flip-Flop problem.

For Generic Algorithm (GA), the following parameters are tuned: population_sizes = [100, 200, 500, 1000, 1500, 2000], mutation_rates = [0.1, 0.2, 0.4, 0.6, 0.8].

From Figure 7, it is observed that GA performs best when population size is 1000. For population size being, GA balance exploration and exploitation.

GA is not sensitive to mutation rate. As mutation rate varies from 0.2 to 0.8, the fitness function only varies from 280 to 288. It is observed that mutation rate of 0.8 does help GA find global optima.



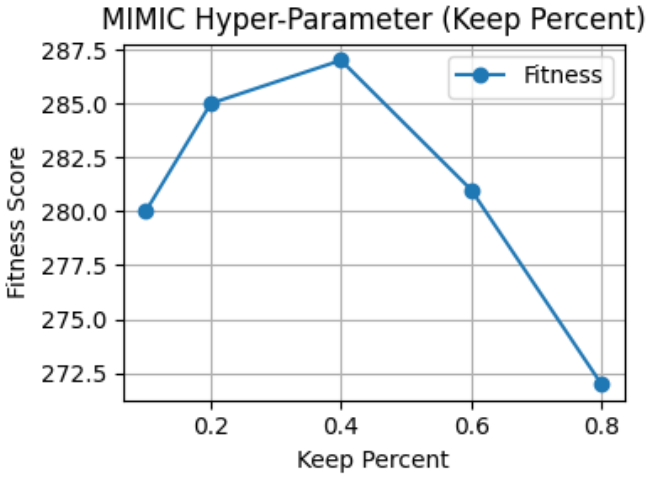
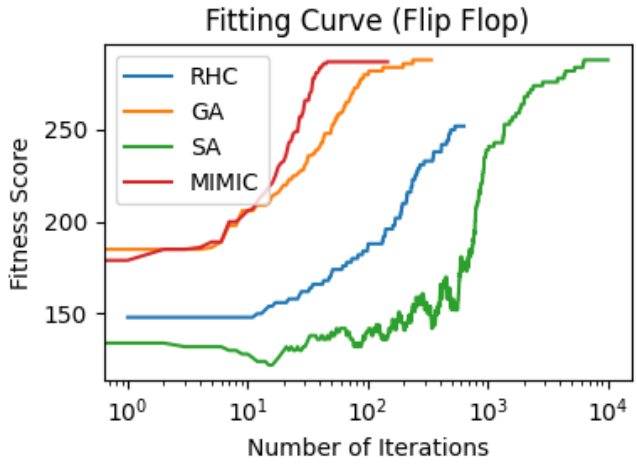


Fig. 8. Hyper-parameter tunning for MIMIC in Flip-Flop problem.



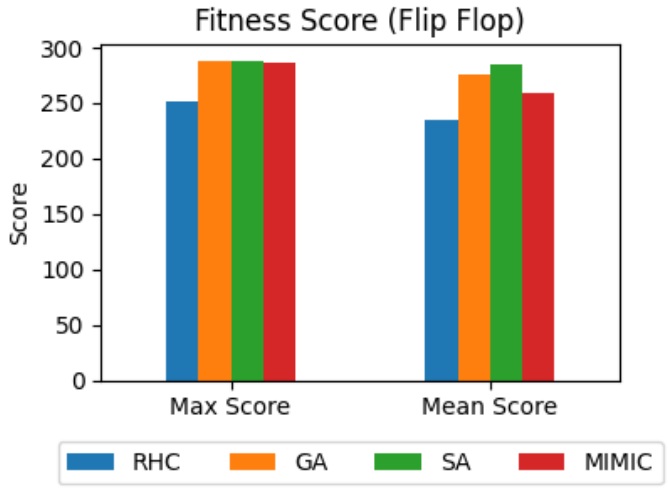


Fig. 9. Fitness score comparison for Flip-Flop problem.

For MIMIC, the following parameters are searched: population_sizes = [100, 200, 500, 1000, 1500, 2000], keep_percent_list=[0.1, 0.2, 0.4, 0.6, 0.8].

From Figure 8 upper plot, it is observed that the fitness scores of MIMIC increase with population size. Higher population size allows the MIMIC to explore more for this problem.

From Figure 8 lower plot, it is observed that the best keep percent for Four Peaks problem is 0.4. Keep percent of 0.4 helps the MIMIC algorithm achieve a balance between exploration and exploitation.

B. Fitness Score Comparison

In Figure 9, the fitness curves of the four algorithms with their best tunned parameters are plotted.

From Figure 9, it is observed that RHC did the worst. RHC is trapped in local optima and this best-case optimum found by RHC is significantly smaller than global optima.

Looking the at number of iterations in Figure 9, MIMIC converges the most quickly of all four algorithms. But the optimum found by MIMIC is slightly less than SA and GA.

In Figure 9, the highest fitness score of GA is the same as that of SA, and GA uses much less iterations to achieve its highest fitness score.

In Figure 9, the SA finds the global optima, but SA took much more iterations than GA to find the global optima. One interesting observation about SA is that the fitness score of SA goes up and down. This is reasonable to SA, since SA has certain probability to explore some locations, where these locations have lower fitness score. The exploration certainly is essential for SA to find global optima. As temperature cools down, SA is mainly exploiting and tries to approach the optimum.

It is noticeable that the mean score of SA is the same as its max score, meaning that SA is not sensitive to initial temperature and SA can find global optimum in all most all

cases, which is why I think SA is best for the Flip-Flop problem, due to its robustness and time efficiency.

C. *Efficiency Comparison*

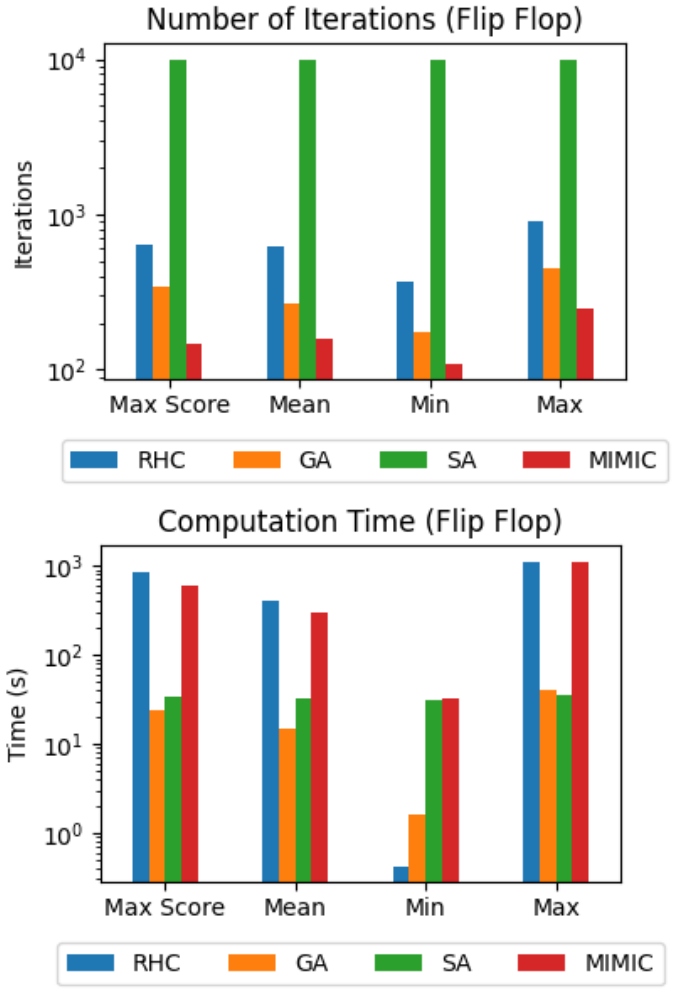


Fig. 10. Efficiency comparison for Flip-Flop problem.

In Figure 10, we observe large variation in iteration number and time for RHC, which makes sense, since RHC is random search. For some cases, RHC gets trapped in local optima then quit, thus uses small number of iteration and time. For some cases, RHC is on the road to find the better optima, thus uses larger number of iteration and time.

MIMIC uses the smallest number of iterations of all four algorithms. But each iteration takes a much longer time than the other three algorithms. Thus, total time is almost the longest of the four algorithms.

The number of iterations of GA is second smallest and the time of each iteration is relatively smallest.

SA uses the greatest number of iterations of all algorithms. But each iteration is quick. Thus, total time consumed by SA is comparable to GA. Thus, considering the effectiveness, the efficiency of SA and GA is the top two of all four algorithms.

IV. Knapsack Problem (Best for MIMIC)

In this section, I document the hyper-parameter search, fitness, and efficiency of the four algorithms for the Knapsack problem.

A. *Hyper-Parameter Search*

For Randomized Hill Climbing, I used the following hyper-parameters: iteration_list = [10000], restart_list = [100], max_attempts = 100.

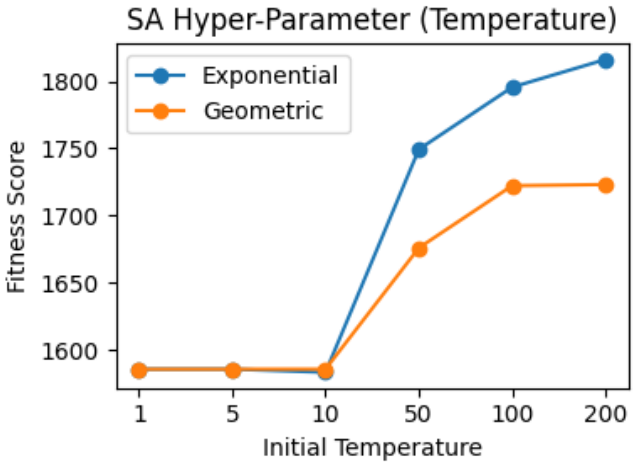
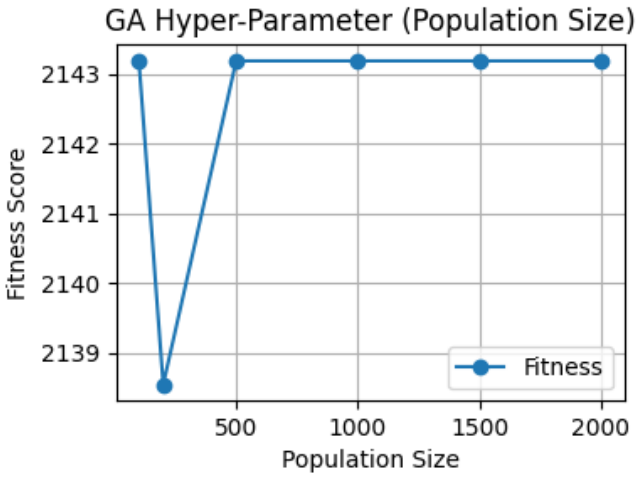


Fig. 11. Hyper-parameter tunning for SA in Knapsack problem.

The Simulated Annealing (SA) hyper-parameters tuned are decay (ExpDecay and GeomDecay) and init_temperature ([1, 5, 10, 50, 100, 200]). For Figure 11, it is observed that ExpDecay is better than GeomDecay for this problem. The best initial temperature is 200 for this problem. For initial temperature being 200, SA can better balance exploration and exploitation.

For Generic Algorithm (GA), the following parameters are tuned: population_sizes = [100, 200, 500, 1000, 1500, 2000], mutation_rates = [0.1, 0.2, 0.4, 0.6, 0.8].



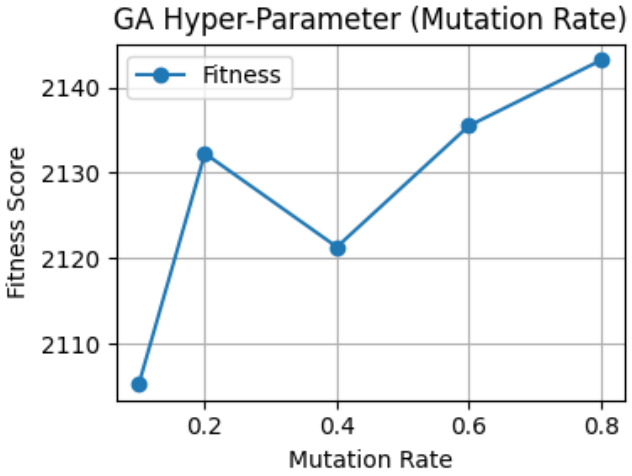
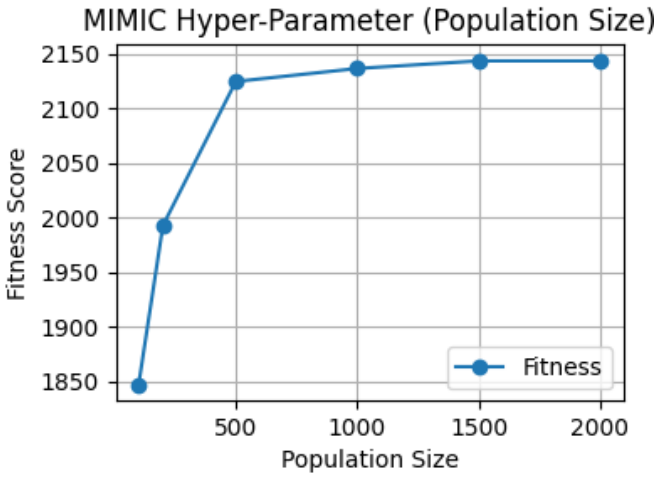


Fig. 12. Hyper-parameter tuning for GA in Knapsack problem.

From Figure 12, it is observed that GA is not sensitive to population size: as population size varies from 10 to 2000, the fitness score varies from 2135 to 2143. GA is sensitive to mutation rate. As mutation rate varies from 0.1 to 0.8, the fitness function only varies from 2050 to 2190. It is observed that mutation rate of 0.8 does help GA find global optima.



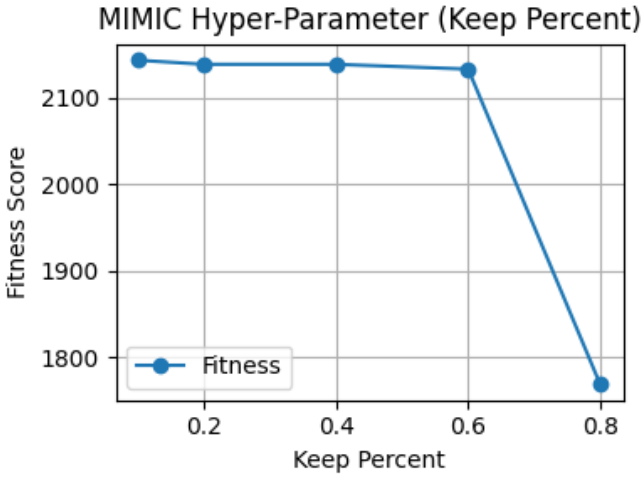


Fig. 13. Hyper-parameter tunning for MIMIC in Knapsack problem.

For MIMIC, the following parameters are searched: population_sizes = [100, 200, 500, 1000, 1500, 2000], keep_percent_list=[0.1, 0.2, 0.4, 0.6, 0.8].

From Figure 13 upper plot, it is observed that the fitness scores of MIMIC increase with population size. Higher population size allows the MIMIC to explore more for this problem. The best population size for MIMIC is 2000.

From Figure 13 lower plot, it is observed that the best keep percent for Four Peaks problem is 0.4. Keep percent of 0.4 helps the MIMIC algorithm achieve a balance between exploration and exploitation.

B. Fitness Score Comparison

In Figure 14, the fitness curves of the four algorithms with their best tunned parameters are plotted.

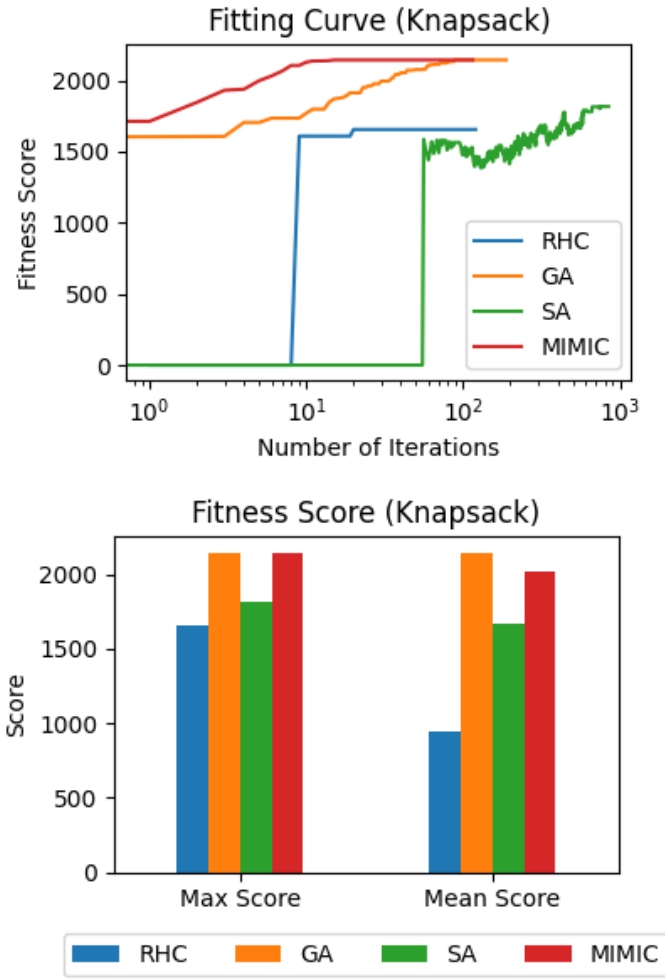


Fig. 14. Fitness score comparison for Knapsack problem.

From Figure 14, it is observed that RHC did the worst. RHC is trapped in local optima and this best-case optimum found by RHC is significantly smaller than global optima.

In Figure 14, SA performs better than RHC. By introducing temperature into SA, SA can explore more than RHC. Thus, SA is more robust for local optima trapping. But due to the randomness of the Knapsack problem, SA till gets trapped in local optima.

In Figure 14, GA and MIMIC both achieves the highest fitness score. But GA find it with much greater number of iteration than MIMIC. This is because MIMIC does well when there is structure beneath, which is the case for the Knapsack problem.

C. *Efficiency Comparison*

Next, we take a close look at the efficiency of the algorithms. More specifically, we will examine the number of iterations and time used in maximization.

In Figure 15, we observe large variation in iteration number and time for RHC, which makes sense, since RHC is random search. For some cases, RHC gets trapped in local optima then quit, thus uses small number of iteration and time. For some cases, RHC is on the road to find the better optima, thus uses larger number of iteration and time.

SA uses the greatest number of iterations of all algorithms. But each iteration is quick. Thus, total time consumed is smallest among the four algorithms.

The number of iterations of GA is second smallest and the time of each iteration is decent.

MIMIC uses the smallest number of iterations of all four algorithms. But each iteration takes a long time. When comparing the time used to solve the Knapsack problem, the time consumed is similar for RHC, GA and MIMIC.

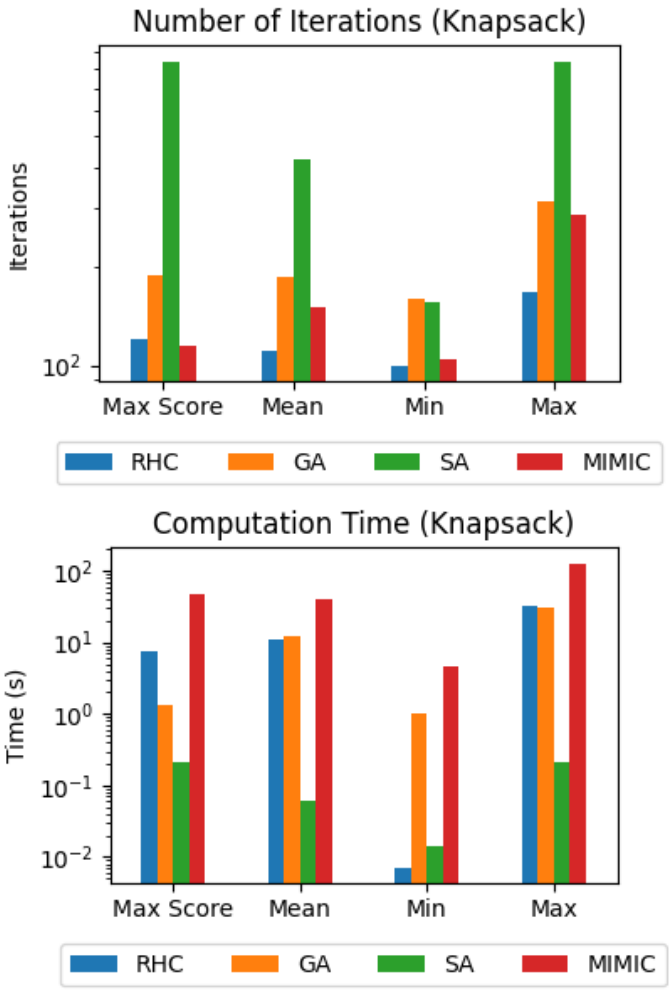


Fig. 15. Efficiency comparison for Knapsack problem.

V. Neural Network (Best for ADAM)

For the study of optimization algorithms for neural network, I used Planar data.

A. *Hyper-Parameter Search*

In project one, the hyper-parameter of the Neural Network has been tuned: 1 hidden and 5 nodes, since the Planar is

simulated data, which is much simpler than real deep-learning problem. The learning rate is 0.05.

In this section, I will focus the hyper-parameter tunning of the optimization algorithms.

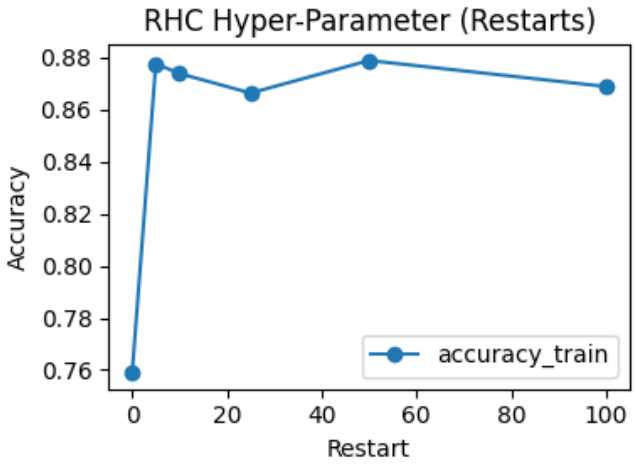


Fig. 16. Hyper-parameter tuning for RHC in Neural Network problem.

For RHC, I tuned the number of restarts from 0 to 100. For Figure 16, it is observed that adding restarts from 0 is helpful to RHC. But once we have 5 restarts, adding more restarts does not further improve the training accuracy.

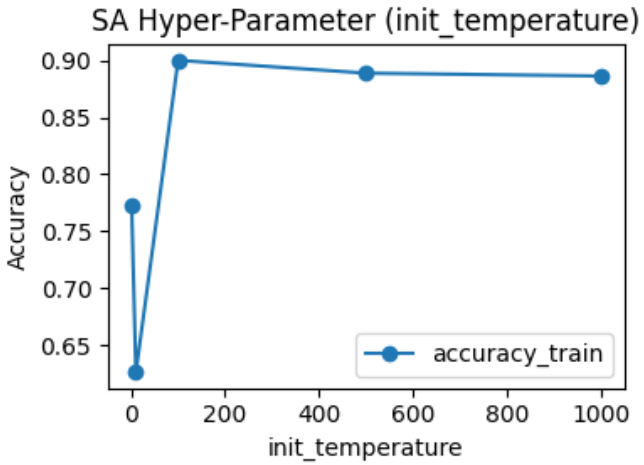


Fig. 17. Hyper-parameter tuning for SA in Neural Network problem.

For SA, I tuned the initial temperature from 1 to 1000. From Figure 17, the best initial temperature for SA in Neural Network of Planar data is 100. Temperature lower than 100 reduces the SA performance significantly, because low temperature will prevent SA exploring. Temperature higher than 100 reduce the SA performance slightly and gradually.

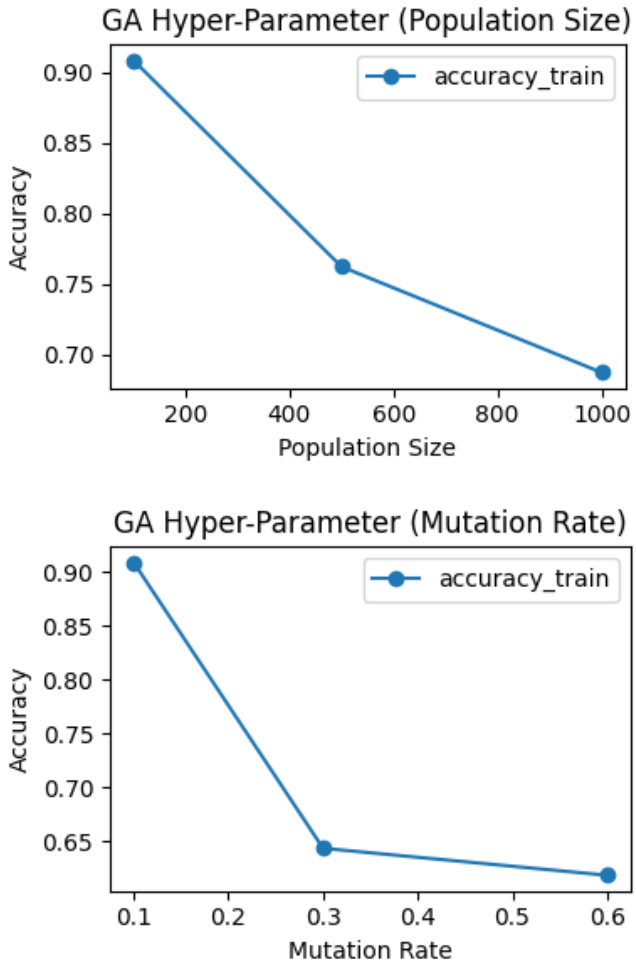


Fig. 18. Hyper-parameter tuning for GA in Neural Network problem.

For GA, I used grid search for population size of [100, 500, 1000] and mutation rate [0.1, 0.3, 0.6]. Figure 18 shows that the training accuracy decreases as training accuracy increases. This is because too large populations size prevents GA to fine tune the weights and gets stabilized, which is essential for later stage optimization for Neural Network.

In Figure 18, the training accuracy decrease as mutation rate increase, which makes sense for GA. Because too large mutation rate prevents GA to fine tune the weights and gets stabilized, which is essential for later stage optimization for Neural Network.

B. Fitness Score Comparison

In Figure 19, I compare for optimization algorithms: ADAM, RHC, SA and GA. ADAM is the algorithm I used in project one and serves as a benchmark. From Figure 19, we can see that ADAM achieved the highest training and testing accuracy among all algorithms. In terms of training accuracy, GA performs the better than SA and RHC. GA has higher testing accuracy than RHC and SA. This makes sense, since the Neural Network with Planar data is relatively simple, and GA can finetune the weights to achieve good performance.

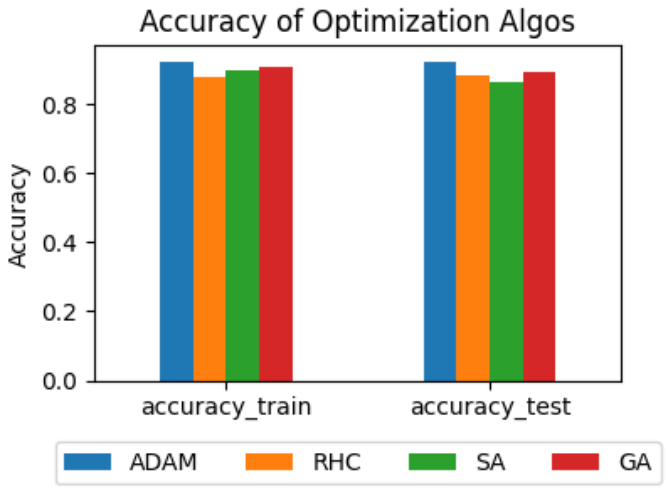


Fig. 19. Fitness score comparison for Neural Network problem.

C. *Efficiency Comparison*

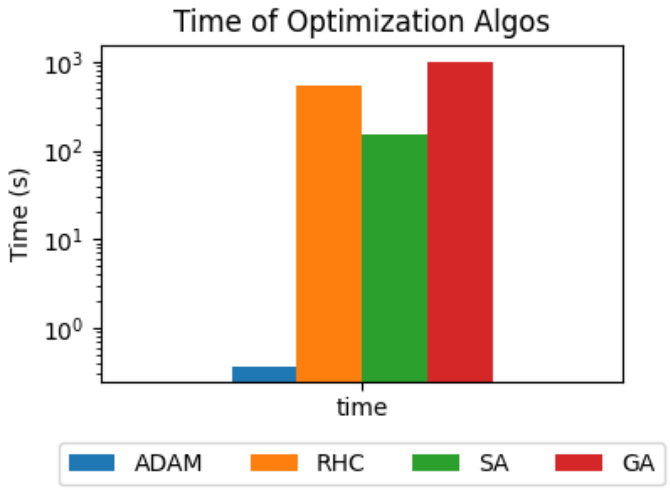


Fig. 20. Efficiency comparison for Neural Network problem.

Figure 20 compares the time efficiency of the optimization algorithms. From Figure 20, it is observed that ADAM uses the smallest amount of time among the four algorithms, demonstrating how efficient it is. For continuous optimization problem, gradient decent really has its advantage.

RHC uses more time than SA, this is because RHC uses restarts.

GA uses more time than RHC and SA, which makes sense. Because each iteration, GA needs to sample from the population and mutate, and then select the best candidates from the mutated population.

Conclusion and Future Work

In project, I applied the Randomize Hill Climbing, Simulated Annealing, Generic Algorithm, and MIMIC to three discrete problems and one Neural Network problem. The behavior, fitness, and efficiency of the four algorithms are compared.

Based on the experiment results, the best algorithm depends on which problem we are trying to solve. Put it another way, each algorithm has its strength and weakness. For example, RHC is good at exhaustive search problem. Each iteration of RHC is fast. But RHC is easy to be trapped in local optima. SA has all the good properties of RHC. Plus, SA explores more than RHC, thus less likely be trapped in local optima than RHC. With the ability to mutate and improve, GA is good at problems with many local optima and problems having structure. MIMIC can give an overview of the problem and enable us to discover the structure of the problem. MIMIC converges fast in terms of the number of iterations, but each iteration of MIMIC is very expensive.

Also, the best algorithm depends on what metrics we choose to measure the algorithms' performance and what we really care about the algorithm: Do we want a goof fitness score? Do we want a small number of iterations? Or do we want a small computation time? Bases on our experiments, if we want a good fitness score, then SA and GA tend to be selected. If we care about computation time, then SA and GA tend to be selected. If we care about the number of iterations, then MIMIC tend to be selected.

References

[1] https://en.wikipedia.org/wiki/Knapsack_problem

[2] https://en.wikipedia.org/wiki/Simulated_annealing

[3] https://en.wikipedia.org/wiki/Generic_programming

[4] P. V. Jeremy DeBonet, Charles Isbell. Mimic: Finding optima by estimating probability densities. In Advances in Neural Processing Systems 1996, 1996.

[5] Charles L. Isbell Randomized Local Search as Successive Estimation of Probability Density