# Comparable analysis of Local Random Search Algorithms

*Zhi Li (zli3037@gatech.edu)

Github Repo

***Abstract:*** **This report presents a comparative evaluation of four local random search algorithms: Randomized Hill Climbing (RHC), Simulated Annealing (SA), Genetic Algorithm (GA), and Mutual-Information-Maximizing Input Clustering (MIMIC). Initially, three classic optimization problems, namely continuous peaks, flip flop, and four peaks, are defined. Subsequently, the aforementioned algorithms are applied to solve these problems. A comprehensive analysis is conducted, examining computation times, iterations, decay rates, problem sizes, populations, and other hyperparameters. Furthermore, the study extends to continuous optimization problems, where the efficacy of local random search techniques is evaluated. Through this investigation, insights into the performance and suitability of these algorithms across discrete and continuous optimization domains are provided.**

## I. Introduction

Local random search algorithms are widely employed in optimization tasks due to their simplicity, versatility, and effectiveness in exploring solution spaces. In this report, we conduct a comparative evaluation of four prominent local random search algorithms: Randomized Hill Climbing (RHC), Simulated Annealing (SA), Genetic Algorithm (GA), and Mutual-Information-Maximizing Input Clustering (MIMIC). These algorithms offer different approaches to exploration and exploitation within solution spaces, making them suitable for a variety of optimization problems.

To begin our investigation, we define and explore three classic optimization problems: continuous peaks, flip flop, and four peaks. These problems serve as benchmarks to assess the performance of the aforementioned algorithms. Each problem presents unique challenges, ranging from rugged landscapes to discrete decision spaces. By applying RHC, SA, GA, and MIMIC to these problems, we aim to elucidate their respective strengths and weaknesses in addressing diverse optimization scenarios.

Our evaluation encompasses various aspects of algorithm performance, including computation times, iteration counts, decay rates, problem sizes, populations, and other hyperparameters. By conducting a comprehensive analysis across these dimensions, we gain insights into how each algorithm behaves under different optimization settings. Moreover, we extend our study to include continuous optimization problems, where the efficacy of local random search techniques is further assessed.

Through this investigation, we seek to provide valuable insights into the performance and suitability of local random search algorithms across discrete and continuous optimization domains. By understanding the strengths and limitations of these algorithms, practitioners can make informed decisions when selecting an appropriate optimization approach for their specific applications. Ultimately, our goal is to contribute to the advancement of optimization methodologies and facilitate the development of more efficient and robust optimization techniques.

## II. Algorithms

### A. Randomized hill climbing (RHC)

RHC starts with an initial solution and iteratively explores neighboring solutions by making random changes. It accepts any neighboring solution that improves the objective function, even if it's only marginally better. It have various variations such as Random Restart Hill Climbing (RRHC).

### B. Simulated Annealing (SA)

Simulated Annealing (SA) is a heuristic optimization algorithm that draws inspiration from the annealing process in metallurgy. This algorithm operates as a hill climbing method with a non-deterministic search for the global optimum. In metallurgy, annealing involves the cooling and freezing of a metal into a minimum-energy crystalline structure. SA mimics this process by iteratively exploring the solution space and occasionally accepting solutions with higher energy (worse fitness) to escape from local optima.

A key parameter in SA is the temperature parameter, often denoted as "T". At higher temperatures, the algorithm has a higher probability of accepting solutions that increase the objective function value, thus facilitating exploration. As the algorithm progresses, the temperature parameter gradually decreases, typically following a predefined cooling schedule. This decrease in temperature reduces the probability of accepting worse solutions, allowing the algorithm to converge towards the global optimum.

While SA theoretically guarantees convergence to the global optimum, its practical effectiveness can vary depending on problem characteristics and parameter settings. One challenge in using SA is determining the rate at which to decrease the temperature parameter. Too rapid cooling may lead to premature convergence to suboptimal solutions, while excessively slow cooling may result in inefficient exploration of the solution space.

In summary, SA offers a powerful approach to optimization by balancing exploration and exploitation. However, careful parameter tuning and selection of an appropriate cooling schedule are crucial for its successful application to optimization problems, as improper settings may impede convergence or lead to suboptimal solutions.

### C. Genetic Algorithm (GA)

Genetic algorithms (GA) emulate natural selection by maintaining a population of candidate solutions and iteratively evolving them through selection, crossover, and mutation. They excel in exploring vast solution spaces, making them suitable for complex optimization problems with multiple optima. However, their effectiveness comes at the cost of computational intensity due to managing a solution population. Furthermore, GAs demand meticulous

parameter tuning, including population size and crossover rate, to achieve optimal performance. Despite these challenges, GAs find widespread applicability across diverse optimization domains, particularly where traditional methods struggle with intricate solution landscapes. Their ability to navigate complex search spaces and uncover diverse solutions makes them a valuable tool for tackling challenging optimization tasks.

### *D. Mutual-Information-Maximizing Input Clustering (MIMIC)*

Mutual-Information-Maximizing Input Clustering (MIMIC) explores the global structure of an optimization problem by employing knowledge of this structure to guide a randomized search through the solution space and refine the estimate of the structure. The algorithm aims to locate optima by estimating probability densities. Initially, MIMIC randomly samples from regions of the input space most likely to contain the optima for the objective function. Subsequently, it utilizes an effective density estimator, which can capture a wide variety of structures on the input space, derived from simple second-order statistics of the data. This approach allows MIMIC to effectively explore the solution space and adapt its search strategy based on the underlying structure of the problem. By leveraging probability densities and sophisticated density estimation techniques, MIMIC offers a powerful approach to optimization, particularly well-suited for problems with complex and multi-modal solution landscapes.

## III. CLASSIC OPTIMIZATION PROBLEMS

### *A. Flip Flop Problem (FFP)*

FFP (Flip-Flop) is a discrete optimization problem that computes the number of alternating bits in a bit string, where any transition from one digit to another counts as one flip. The goal is to minimize the number of flips between consecutive bits in a binary string. The ideal fitness bit string would consist entirely of alternating digits. Mathematically, the fitness function $f(S)$ for a binary string S of length N can be expressed as:

$f(S)$ = N - number of flips in S

The figure 1 shows the comparable analysis of four algorithms on Flip Flop Problem. As we can see from the RHC subplot, increasing the number of random restarts generally improves fitness, because it enhances the algorithm's exploration capability. Also the fitness versus iteration curve converged fast at relative low fitness value in the fitness curve subplot, as the algorithm might stuck at local optima.

As for Simulated Annealing, the author compared three type of decay scheduling: exponential, geometric, and arithmetic. As we can see from the subplot of SA, the three decay schedules influences the trade-off between exploration and exploitation. Exponential decay has the most steep slope as it encourages early exploration, and it slows down later as it focus too much on exploitation and eventually stuck at the local optima. Geometric decay offers a more balanced exploration and exploitation tradeoff throughout entire iterations, as the decay rate becomes more appropriate for this problem. Arithmetic decay does the similar performance at the early stage, but fluatuates on the late stage and fails to converge, it is because the decay rate is always consitent and is large so it keeps skipping the optima.

For GA, the various combination of mutation probabilities (0.1, 0.3) and population sizes (100, 200, 500) are considered. As shown in GA Analysis subplot from Figure 1, the mutation probability and population size are charged. It is noted that, in the early stage, higher mutation probabilities leads to higher fitness but slower convergence due to more exploration, while larger population sizes facilitate better exploration and faster convergence. In the late stage, the mutation prob of 0.1 and popuation size of 200 reach the best tradeoff between exploration and exploitation in this case.

For MIMIC, the author explores different combinations of keep percentages (10% and 30%) and population sizes (100, 200, 500). As shown in MIMIC Analysis subplot from Figure 1, the author considers changing keep percentages and population size. It is seen that the larger population sizes are prone to find global optima as it allows for a more comprehensive exploration of the solution space. The smaller keep percentage lead to more exploitation and faster convergence, because only smaller portion of the top-perfomace are preserved and help focus on refining and exploiting the best solutions found so far.

For comparison across the four algorithms, the author plots fitness vs iteration, fitness vs problem size and computatinal time vs problem size curves for the above four algorithms. As seen in the last three subplots of Fig.1, all four algorithms increase the fitness with the iterations or problem sizes increase. Let make more detailed analysis. For number of iterations, MIMIC required less iterations to converge, because it explores the global structures, and builds probabilistic model of the solution space. GA suprisingly converges at local optima at early stage, it is attributed to insufficient exploration with smaller crossover and mutation propabilities. RHC require fewer iterations as it focuses on local search with less emphasis on global exploration, and it stucks at local optima at early stage. SA involves a moderate number of iterations as it gradually cools down the temperature to explore the solution space efficiently. In terms of problem size, all four algorithms increase the fitness with problem size increased. MIMIC has highest fitness with the increase of problem size, as a bigger problem size lead it to fully exploring the global sturcture. RHC has lowest performace as it does not do global exploration, large sizes may result in stucking in local optima. As for computational time-iteration curves, GA demonstrates slower convergence compared to RHC and SA, its computational time-iteration curve initially rise as the algorithm initializes and evolves the population. Subsequently, the curve exhibits a fluctuation as the algorithm explores and refines solutions through generations. MIMIC's curve shows slower convergence, especially with the problem size increasing, because it iteratively refines the probability distribution. RHC and SA are not sensitive to problem size, as both are focus on local search with less exploration without learning the global structures.

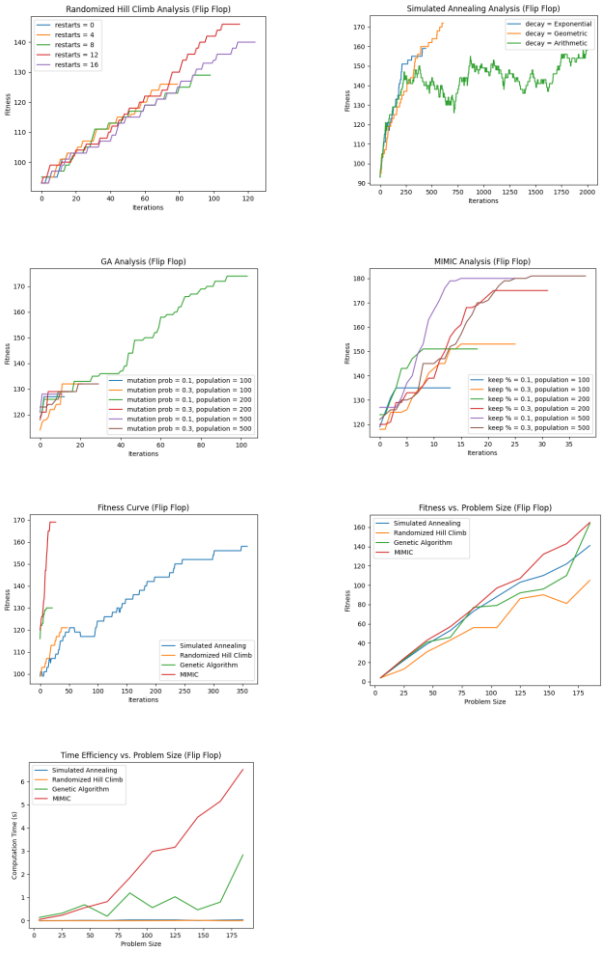



Fig.1 Comparable Analysis of Four Algorithms on Flip Flop Problem.

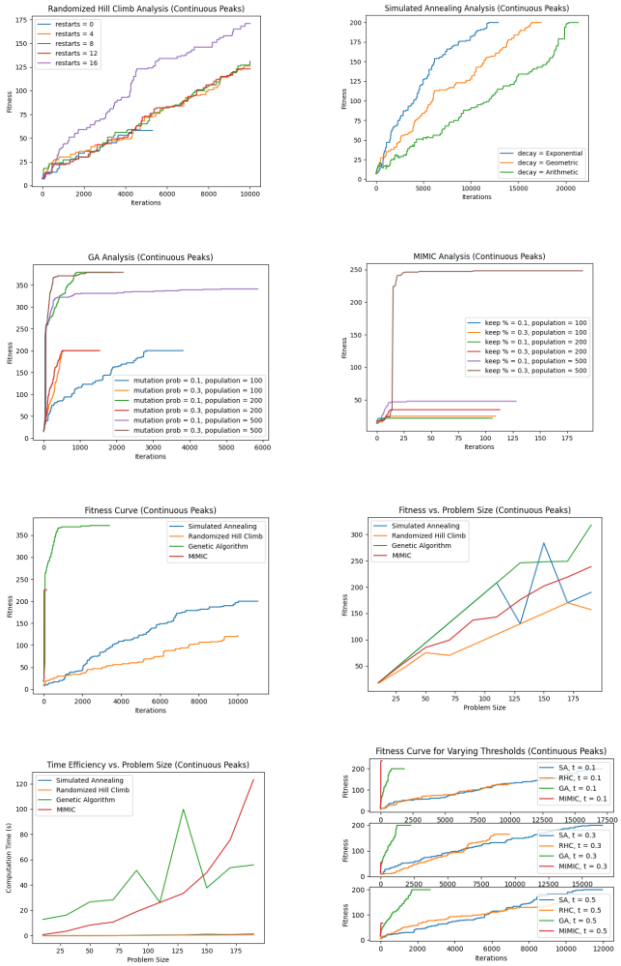



Fig.2 Comparable Analysis of Four Algorithms on CPP

## *B. Continuous Peaks Problem (CPP)*

Continuous Peaks optimization problem involves finding a binary string with a specified number of consecutive 1s within a search space characterized by continuous plateaus and discrete regions. The fitness function evaluates candidate solutions based on the presence of consecutive 1s, with the goal of finding the global optimum with the highest fitness value.

Fitness function for Continuous Peaks optimization problem is the fitness[1] of an n-dimensional state vector x, given parameter T, as:

$$Fitness(x, T) = \max(max\_run(0, x), max\_run(1, x)) + R(x, T)$$

where:

- $max\_run(b, x)$ is the length of the maximum run of b's in $x$;
- $R(x, T) = n$, if $(max\_run(0, x) > T$ and $max\_run(1, x) > T)$; and
- $R(x, T) = 0$, otherwise.

The figure 2 shows the comparable analysis of four algorithms on Continuous Peaks Problem. As it is obviously seen from the RHC subplot, increasing the number of random restarts significantly improves fitness, because it enhances the algorithm's exploration capability.

As for Simulated Annealing, we can see from the subplot of SA, unlike the flip-flop problem, exponential decay has the best performance, fastest convergence to the global optima. It has the most steep slope as it encourages early exploration, a more balanced exploration and exploitation tradeoff. The main reason is that the CPP's fitness landscape contains continuous regions of increasing fitness values, making it more amenable to exploration. Exponential decay allows SA to explore the continuous regions more effectively. Both geometric decay and arithmetic decay find the global optima.

As shown in GA Analysis subplot from Figure 2, higher mutation probabilities and larger population size lead to higher fitness and fast convergence, which is different from the Flip-Flop Problem. The case with mutation probability of 0.3 and population of 500 has best exploration and exploitation tradeoff.

As shown in MIMIC Analysis subplot from Figure 2, similar to the result of flip-flop problem, the larger population sizes are prone to find global optima as it allows for a more comprehensive exploration of the solution space. MIMIC is the most robust and stable algorithm for both problems, which might be because it uses knowledge of the gloabel structure to guide a randomized search through the solution space and to refine the estimate of the structure.

For further comparison, as depicted in the last four subplots of Fig. 2, all four algorithms demonstrate an increase in fitness with the rise in iterations or problem sizes. Regarding the number of iterations, MIMIC requires the fewest iterations to converge, leveraging its capacity to explore global structures and construct a probabilistic model of the solution space. Surprisingly, GA achieves convergence at the global optima early on. Conversely, RHC demands fewer iterations, focusing primarily on local search with minimal emphasis on global exploration, yet it tends to become trapped in local optima at an early stage. SA entails a moderate number of iterations, gradually cooling down the temperature to explore the solution space effectively. Concerning problem size, all four algorithms exhibit an enhancement in fitness with increasing problem size. MIMIC achieves the highest fitness levels as problem size expands, attributed to its capability to fully explore the global structure. Conversely, RHC displays the lowest performance due to its limited global exploration, which may lead to stagnation in local optima with larger problem sizes. In terms of computational time-iteration curves, GA showcases slower convergence compared to RHC and SA. Initially, its computational time-iteration curve rises as the algorithm initializes and evolves the population. Subsequently, fluctuations are observed as the algorithm explores and refines solutions across generations. MIMIC's curve illustrates slower convergence, particularly with increasing problem size, attributable to its iterative refinement of the probability distribution. Notably, RHC and SA exhibit insensitivity to problem size, as both predominantly focus on local search with minimal exploration, lacking the capacity to learn global structures. As seen in the last subplot of Fig.2, lower thresholds result in lower fitness values because they require fewer consecutive bits to meet the criteria for a high fitness value. This leads to faster convergence as the algorithm can more readily identify and exploit shorter continuous peaks, prioritizing solutions that require less exploration to meet the threshold.

### *C. Four Peaks Problem (FPP)*

The fitness function for the Four Peaks problem is defined as the maximum of two components: the number of consecutive 1s and the number of remaining elements not part of the consecutive 1s. The fitness [1] of an n-dimensional state vector x, given parameter T, as

$$Fitness(x, T) = \max(tail(0, x), head(1, x)) + R(x, T)$$

where:

- $tail(b, x)$ is the number of trailing b's in $x$;
- $head(b, x)$ is the number of leading b's in $x$;
- $R(x, T) = n$, if $tail(0, x) > T$ and $head(1, x) > T$; and
- $R(x, T) = 0$, otherwise.

Figure 3 presents a comparative analysis of four algorithms applied to the Four Peaks Problem (FPP). Increasing the number of random restarts in the RHC subplot notably enhances the algorithm's exploration capability, leading to improved fitness.

In the SA subplot, exponential decay outperforms other decay schedules, showcasing the fastest convergence to the global optimum. The steep slope associated with exponential decay facilitates early exploration, achieving a balanced exploration-exploitation tradeoff.

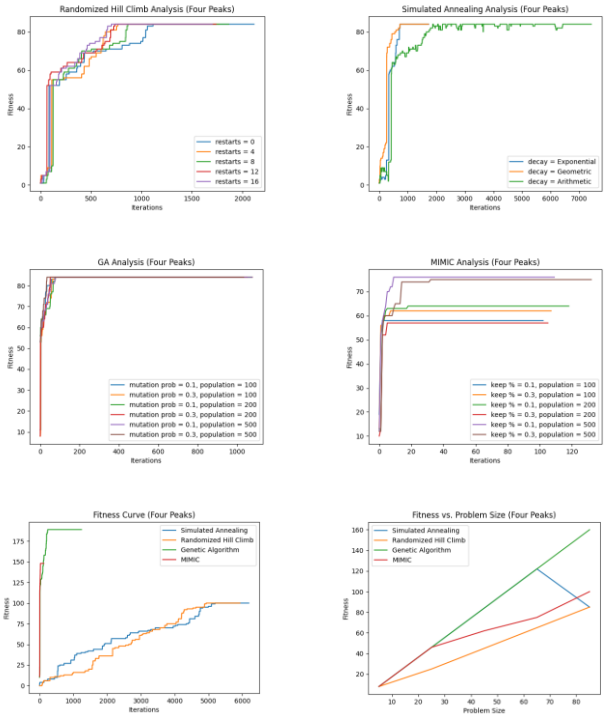
Additionally, the GA Analysis subplot reveals that higher mutation probabilities and larger population sizes lead to enhanced fitness and faster convergence. This differs from the Flip-Flop Problem, with the optimal tradeoff observed at a mutation probability of 0.3 and a population size of 500.

Similarly, in the MIMIC Analysis subplot, larger population sizes facilitate comprehensive exploration, contributing to the algorithm's robustness and stability across both problems. MIMIC leverages knowledge of the global structure to guide exploration and refine estimates effectively.

Moreover, regarding iterations and problem sizes, all algorithms demonstrate increased fitness with rising iterations or problem sizes. MIMIC requires the fewest iterations to converge due to its capacity to explore global structures. Conversely, RHC tends to become trapped in local optima early on, while SA entails a moderate number of iterations, gradually exploring the solution space. As for problem size, MIMIC achieves the highest fitness levels due to its ability to fully explore global structures.

In terms of computational time-iteration curves, GA showcases slower convergence compared to RHC and SA, attributed to its initialization and evolution process. Conversely, MIMIC exhibits slower convergence, particularly with larger problem sizes, as it iteratively refines the probability distribution.

Lastly, lower thresholds result in lower fitness values, facilitating faster convergence by prioritizing solutions requiring less exploration to meet the threshold.



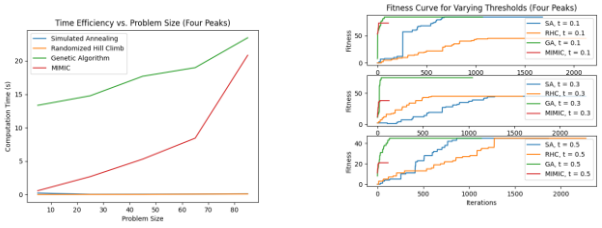
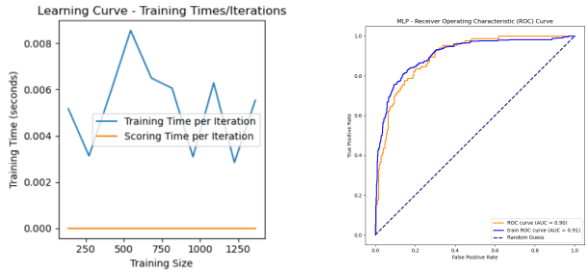



Fig.3 Comparable Analysis of Four Algorithms on FPP

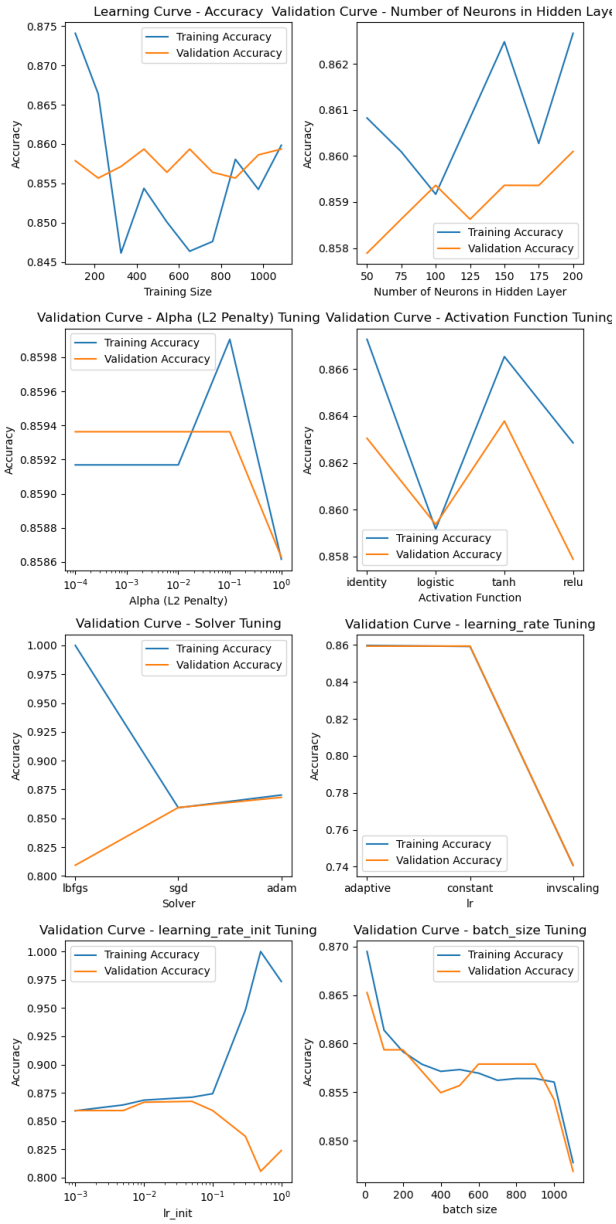
## IV. NERUAL NETWORK WEIGHT OPTIMIZATION PROBLEM

Let recall the first assignment (A1) for the identification of disadvantaged communities. It contains 1698 samples and 16 features, we want to classify the given communities are disadvantaged or not. So far in the previous assignment, various activations, batch size, number of cells, learning rates and optimizers are investigated as shown in Figure 4 below.



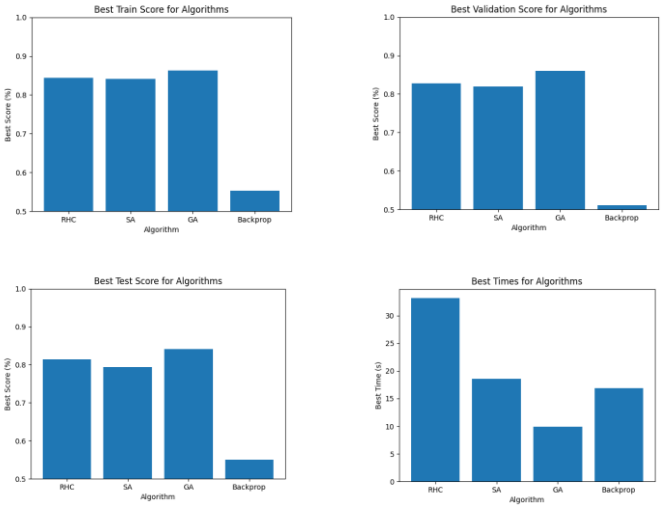

```
Test AUC: 0.90
Train AUC: 0.91
Test Accuracy:  0.85
Train Accuracy: 0.87
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.94      0.90       255
           1       0.75      0.58      0.65        85

    accuracy                           0.85       340
   macro avg       0.81      0.76      0.78       340
weighted avg       0.84      0.85      0.84       340
```

Fig.4 A1: Performance of NNs (Best Hyperparameters: {'activation': 'logistic', 'alpha': 0.01, 'batch_size': 100, 'hidden_layer_sizes': (150,), 'learning_rate': 'adaptive', 'learning_rate_init': 0.1, 'solver': 'sgd'})

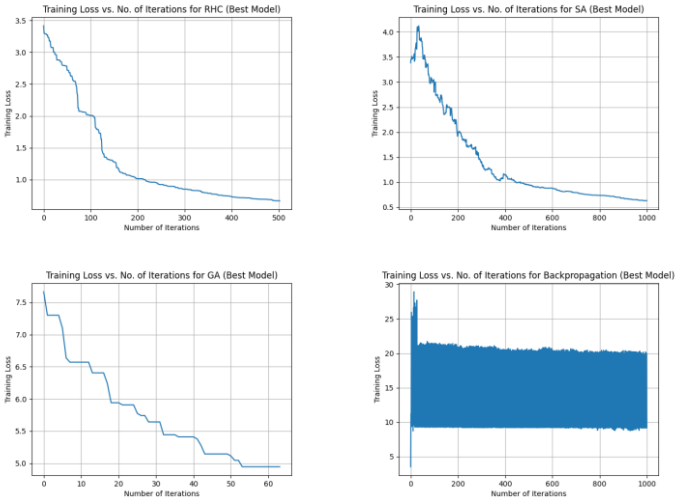



As we can see, GA has the greatest score, outperforming RHC, SA. Backprop does the worst. For this dataset, the GA firstly explores the solution space most effectively and then fully exploits the solution until convergence. The runtime for GA is also the least, because the smaller population size(20) helps explore the space fast and effectively. It accurately finds the domain of the global optima. Then the small mutation probability (0.01) makes it fully exploit the best solutions, leading fast to the optimum. Based on these observations, the solution space for this neural network on the dataset is likely smooth and contains fewer local optima.

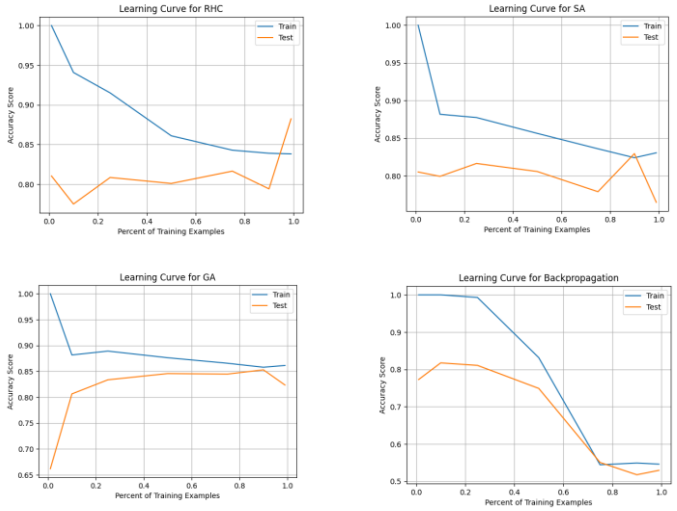
Let see from the other aspect from comparing RHC and SA. RHC slightly outperforms the SA (see Figure 5 a). Because the solution space is smooth with less local optima, RHC just go straight and faster to the global optima, but SA wasted some runtime exploring and then leads to the optima as we can see from the loss curves in Figure 5 b.



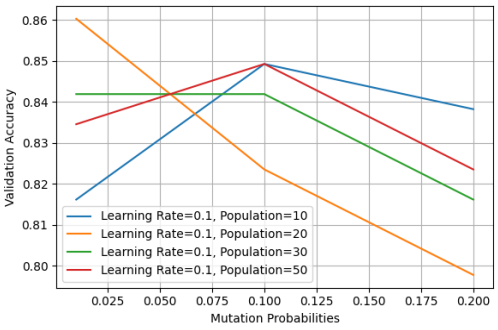

*a) Best models performance*



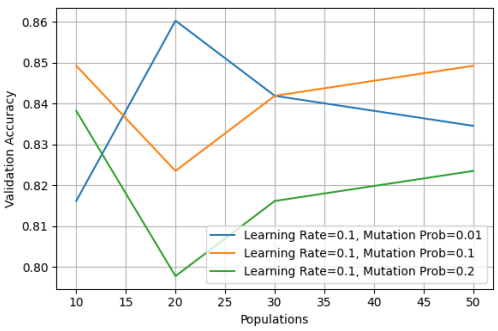

*b) Training loss curve*




*c) Learning curve*




*d) GA accuracy vs mutation prob validation curve*




*e) GA accuracy vs population size validation curve*

Fig.5 Performance of RHC, SA, GA, SGA on NNs

As for Backpropagation, it has the worst score because it has a hard time converging as shown in the loss-iteration curve(see Figure 5 b). The learning rate might be too large for converging. However, it is noted from A1 that the best optimizer is the SGD with the learning rate. The reason is mainly attributed to the adoptive learning rate schedule used in A1. The adoptive learning rate schedule can adjust the learning rate dynamically based on the characteristics of the optimization problem and the training progress. It keep adjusting the learning rate during both the exploration and exploitation phases.

Regarding the learning curves(see Figure 5 c), we can see from the Fig.5, the ideal training size for RHC, SA and GA is around 0.8. they exhibit more robust performance across a range of training sizes. GA is the most robust because it is a population-based optimization approach, which allows for parallel search and evolutionary principles. GA maintains a population of candidate solutions and evolves them over multiple generations. This allows GA to explore a diverse set of solutions simultaneously, regardless of the training size. Additionally, GA's parallel search capability allows it to efficiently search across multiple candidate solutions in parallel. By leveraging evolutionary principles, the diversity introduced through mutation and crossover operations ensures that GA can explore a wide range of potential solutions, further enhancing its robustness. However, for backpropagation, it is sensitive to training size, the small leads to the significant overfitting, the large leads to underfitting. Because when the training size is too small, the model may memorize the training data (overfitting), resulting in poor generalization to unseen data. Conversely, when the training size is too large relative to the model's capacity, the model may struggle to capture the underlying patterns in the data (underfitting).

Let focus on validation curves for GA(see Figure 5 d,e), the different combinations of population size and mutation probabilities are investigated. Initially, we experimented with larger population sizes, observing comparable or worse scores but significantly increased runtime. Consequently, we shifted our focus towards smaller population sizes and mutation probabilities. On further examination, we found that smaller population sizes coupled with high mutation probabilities yielded lower accuracy. This can be attributed to the higher likelihood of disrupting potentially good solutions with the frequent introduction of random changes. Conversely, when employing smaller mutation probabilities, it is sensitivity to population size. Larger population sizes tended to prolong convergence, while smaller population sizes were prone to becoming trapped in local optima.

## V. CONCLUSION

In conclusion, after comparing 4 algorithms to search for solutions. It is found that each algorithms has its strengths and weaknesses.

For classic problems like Flip Flop and Continuous Peaks, we saw that some algorithms are better at finding solutions than others. For example, some algorithms might be good at finding local solutions, while another might be better at exploring the entire space.

| Comparison | RHC | SA | GA | Backpropagation |
|---|---|---|---|---|
| Exploration vs. Exploitation | RHC typically focuses more on local search, which might lead to faster convergence but could get trapped in local optima. | SA introduces a temperature parameter that controls the balance between exploration and exploitation. | GA uses a population-based approach, allowing for more diverse exploration of the solution space through crossover and mutation operations. | Backpropagation updates weights based on gradients computed using backpropagation, balancing between exploring different paths and exploiting gradients for convergence. |
| Convergence Speed | RHC may converge quickly but might get stuck in local optima. | SA often has a slower convergence rate but can escape local optima by accepting worse solutions with a certain probability. | GA typically requires more iterations to converge due to its population-based nature and the need to explore and exploit diverse solutions. | Backpropagation often exhibits a gradual decrease in loss as it updates weights based on gradients, contributing to convergence. |
| Parameter Sensitivity | RHC doesn't have many parameters to tune, but initial conditions can affect its performance. | SA requires tuning the initial temperature and cooling schedule. | GA requires tuning parameters such as population size and mutation rate. | Backpropagation requires tuning parameters like learning rate and regularization strength to balance convergence speed and stability. |
| Problem Complexity | RHC may struggle with complex problems and high-dimensional spaces. | SA can handle complex problems but might need careful tuning. | GA is versatile but may not be efficient for highly complex problems. | Backpropagation is commonly used for training neural networks on complex problems with high-dimensional input spaces. |
| Training Size | RHC may not be affected significantly by training size as it primarily focuses on local search. | SA performance may improve with larger training sizes due to better exploration of the solution space. | GA often benefits from larger training sizes as it enhances diversity in the population and improves convergence. | Backpropagation typically requires large training sizes to effectively learn complex patterns in data and avoid overfitting. |
| Computational Time | RHC tends to have lower computational time compared to other algorithms due to its local search nature. | SA computational time can vary depending on the cooling schedule and the acceptance probability threshold. | GA computational time increases with the population size and the complexity of the problem. | Backpropagation computational time can be high, especially for deep neural networks and large datasets, due to iterative gradient calculations and weight updates. |
| Robustness | RHC may lack robustness as it's prone to getting stuck in local optima. | SA exhibits robustness against local optima due to its probabilistic acceptance of worse solutions. | GA can be robust due to its population-based approach, which allows for diversity and exploration of multiple solutions. | Backpropagation can be robust against noise in data but may suffer from overfitting if not regularized properly. |

When looking at solving problems with neural networks, the Genetic Algorithm (GA) worked the best in this case. It was good at finding the tradeoff between exploration and exploitation. It was also robust to different sizes of training data. On the other hand, Backpropagation was sensitive to the training size.

The detailed comparison is summarized in 7 key aspects and listed in the table above.

In terms of exploration versus exploitation, the algorithms exhibit varying strategies. RHC predominantly exploits local search, potentially facilitating rapid convergence but risking trapping in local optima. SA introduces a temperature parameter to balance exploration and exploitation, while GA diversifies exploration through a population-based approach. Backpropagation, employed in neural networks, balances between exploring different paths and exploiting gradients for convergence.

Convergence speed, RHC may achieve quick convergence but is susceptible to getting stuck in suboptimal solutions. In contrast, SA typically converges slower but can overcome local optima by probabilistically accepting inferior solutions. GA often necessitates more iterations for convergence due to its population-based nature and the need for diverse exploration. Backpropagation typically exhibits a gradual decrease in loss during convergence, essential for training neural networks effectively on complex datasets.

Furthermore, parameter sensitivity varies among the algorithms. RHC has fewer parameters but may be sensitive to initial conditions. SA and GA require careful tuning of parameters such as temperature, population size, and mutation rate. Backpropagation demands tuning of learning rate and regularization strength to ensure convergence speed and stability.

In terms of problem complexity, RHC may struggle with intricate problems and high-dimensional spaces. SA demonstrates efficacy in handling complex problems, albeit requiring meticulous tuning. GA is adaptable but may not excel in highly complex scenarios. Backpropagation proves effective for complex problems with high-dimensional input spaces, commonly employed in training neural networks.

Additionally, for robustness, RHC may lack robustness due to it is prone to local optima. SA exhibits resilience owing to its exploration. GA showcases robustness through its population-based exploration. Backpropagation can withstand noise in data but may suffer from overfitting, or underfitting if learning rate is large.

## REFERENCES


[1] Hayes, G. (2019). mlrose: Machine Learning, Randomized Optimization and SEarch package for Python. https://github.com/gkhayes/mlrose. Accessed: day month year.

[2] Introduction, Implementation and Comparison of Four Randomized Optimization Algorithms, https://medium.com/@duoduoyunnini/introduction-implementation-and-comparison-of-four-randomized-optimization-algorithms-fc4d96f9feea.