Comparative Analysis of Random Search Techniques on Two Optimization Domains

1st Zixin Feng
Georgia Institute of Technology
Atlanta, USA
zfeng305@gatech.edu

***Abstract*—In this paper, I employed 3 random search techniques: random hill climbing, simulated annealing and a genetic algorithm to solve 2 classical problems – N Queens and 4-peaks. My objective was to thoroughly optimize the hyper parameters for each algorithm and select the best model based on best fitness over clock time, best fitness over iterations, best fitness over problem size and function evaluations. Due to special feature of each problem, some algorithms are supposed to perform better and above hypothesis will be tested and verified in the later analysis. I also plan to apply three algorithms with Gradient Descent to a neural network classification problem using the hospital dataset explored in Assignment1. Through this process, I aim to validate my hypothesis by comparing the performance of these algorithms and selecting the optimal model based on their performance metrics.**

I. INTRODUCTION TO RANDOM OPTIMIZATION PROBLEMS

A. N-Queens

The N-Queens problem is a classical chess game, where players are asked to put N queens on an N times N board and no two queen are in the same raw, column or even diagonal. Bell and Stevens (2009) provide "a survey of known results and research areas for n-queens" which illustrates different algorithms applicable to the N-Queens. I find this problem fascinating because it resembles a common game many of us played in our childhood. Solving this familiar challenge using random optimization techniques is particularly intriguing and appealing.

B. Travelling Sales

The Traveling Salesman Problem (TSP) involves finding the shortest possible route for a salesman to visit a set of cities. The salesman must start from an initial city, visit each city exactly once, and return to the original city. Each city's location is defined by its coordinates (X, Y) I find this problem interesting because it's very classical, usually used as the benchmark problems in optimization worlds. "Due to its complexity, various heuristic and metaheuristic algorithms are used to find near-optimal solutions. These include Genetic Algorithms, Simulated Annealing, Ant Colony Optimization (ACO), and others.".

C. Hypothesis

- Simulated Annealing is expected to outperform both Genetic Algorithm and Random Hill Climbing in solving N-Queens problems. When even minor adjustments to the current state can lead to two queens threatening each other, Random Hill Climbing may prematurely converge to a local minimum. In contrast, Simulated Annealing's exploration capability enables it to escape local optima and continue searching for better solutions. Additionally, Simulated Annealing's simplicity allows for effective management of its cooling schedule, leading to better results compared to Genetic Algorithm, which may require more iterations and computational time.
- Genetic Algorithm is expected to excel in solving Traveling Salesman Problems (TSP), particularly as the problems become more complex with increasing problem size. This advantage stems from the inherent ability of "heuristic and metaheuristic algorithms, including Genetic Algorithm, to effectively tackle intricate problems like the TSP".
- The Genetic Algorithm is expected to excel in solving complicated problems, but its training time might be longer due to the need for more iterations to achieve convergence.

II. N-QUEENS AND SIMULATED ANNEALING

A. Introduction of Simulated Annealing

B. Hyper parameters tuning

1) Decay Rate: In Simulated Annealing, a high temperature encourages broad exploration of the solution space, while a low temperature focuses the search on a local and smaller area. The decay rate defines the speed of the cooling schedule. If the decay rate is too high, the algorithm quickly shifts to exploitation, potentially getting stuck in local minima due to insufficient exploration. If the decay rate is too low, the algorithm performs a more thorough search, but this prolonged exploration can result in poor time efficiency.

I have tested different decay rate values, among which 0.95 is the best over all problem sizes as shown in the table below:

When the problem size is small, the impact of different decay rates on the performance of Simulated Annealing is minimal. However, as the problem becomes more complex, higher decay rates tend to cause the algorithm to converge prematurely to local optima. For instance, in this context, better fitness corresponds to a lower fitness score. When the problem size is 4, both decay rates of 0.95 and 0.99 can find

Decay Value	Problem Size	Best Fitness	Duration
0.95	4	0	0.0194s
0.99	4	0	0.0193s
0.95	6	0	0.033
0.99	6	1	0.0615
0.95	10	0	0.0194
0.99	10	1	0.0493
0.95	12	0	0.0584
0.99	12	1	0.0352

TABLE I
COMPARISON OF DIFFERENT DECAY RATE IN N-QUEENS PROBLEM

the global minimum within a similar time frame. In contrast, for problem sizes of 10 and 12, the best fitness achieved with a decay rate of 0.99 is higher than that achieved with a decay rate of 0.95, indicating that the slower decay rate (0.95) is more effective in finding better solutions for larger, more complex problems.

C. *Performance of various algorithms*

- As the problem becomes more complex, the quality of the solution tends to deteriorate. According to Figure 1, for small problem sizes, both Genetic Algorithms and Simulated Annealing consistently find the global optimum, with their performance overlapping in the graph. However, as the problem size increases to 10 and 12, none of the three algorithms can achieve the global optimum. For problem sizes larger than 12, the fitness scores increase further, indicating a decline in solution quality.

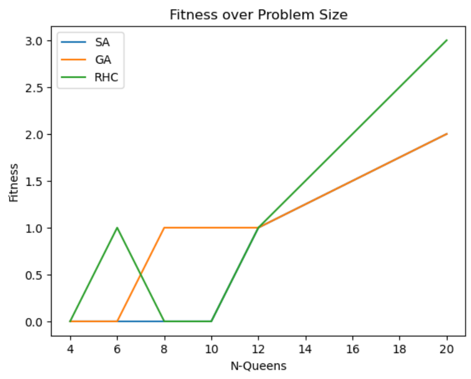

Fig. 1. Fitness/Problem Size

- Due to the simplicity of their algorithms, Simulated Annealing and Randomized Hill Climbing require significantly less time compared to Genetic Algorithms. As clearly shown in Figure 4, as the problem size increases, the duration for Simulated Annealing and Random Hill Climbing increases only slightly, whereas the duration for Genetic Algorithm increases substantially.
- Simulated Annealing requires the fewest function evaluations among the three algorithms. As shown in Figure 2, Simulated Annealing only needs 250,000 function evaluations, while the other two algorithms require more than 300,000 function evaluations each. This explains why

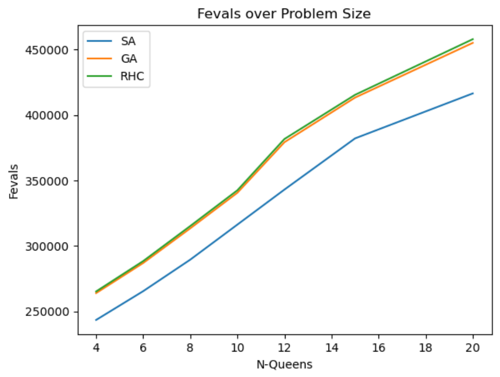

Fig. 2. Function Evaluations in N-Queens Problem

Simulated Annealing has a shorter duration compared to the other two algorithms, which need more time due to the higher number of function evaluations. Simulated Annealing requires fewer function evaluations because it operates from a single solution point rather than multiple combinations. Each time Simulated Annealing explores solutions, it evaluates only one neighbor, unlike Genetic Algorithms, which evaluate multiple neighbors simultaneously.

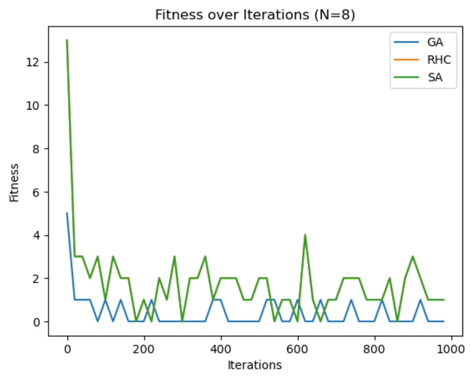

Fig. 3. Fitness/Iteration in N-Queens Problem

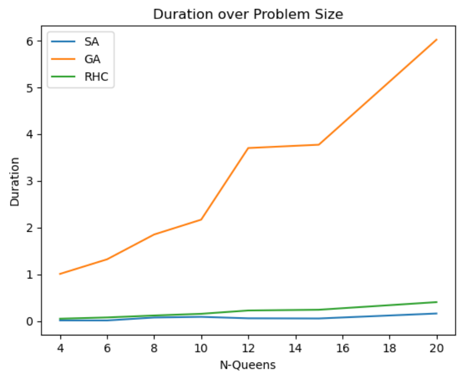

Fig. 4. Duration Comparison in N-Queens Problem

- In general, better solutions are found with more iterations. In Figure 3, which presents an 8-Queens problem and measures fitness over different iterations, we can clearly see that solutions improve as iterations increase from 0 to 100. However, Genetic Algorithms typically find better solutions than Simulated Annealing with fewer iterations. In Figure 3, the fitness score for Genetic Algorithm is consistently lower than that for Simulated Annealing and Randomized Hill Climbing, which overlap. This is because Genetic Algorithm starts with a population of multiple solutions, allowing it to explore multiple solutions simultaneously and perform crossover, whereas Simulated Annealing and Random Hill Climbing operate from a single starting point.

D. Best Algorithm to solve N-Queens Problem – Simulated Annealing

In summary, Simulated Annealing emerges as the most effective algorithm for solving the N-Queens problem, exhibiting shorter runtime, fewer function evaluations, and achieving lower fitness scores compared to other algorithms considered

III. Travelling Salesman Problem and Genetic Algorithm

This section uses Simulated Annealing, Genetic Algorithm and Random Hill Climbing to solve travelling salesman problem and identify Genetic Algorithm as the best algorithm considering the model performance, regardless of time.

A. Hyper parameters tuning

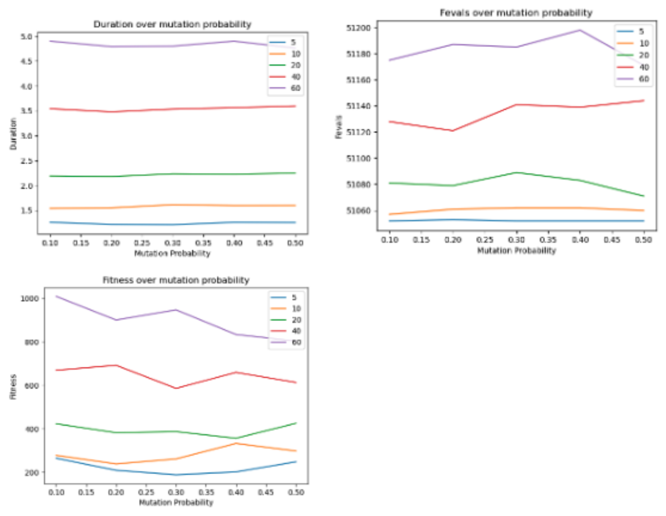

Fig. 5. Mutation Probability tuning Test

1) Mutation Probability: During Genetic Algorithm's algorithm, A population of candidates are selected and recombined. Then mutation of the offspring will be introduced. When the mutation probability is very large, more new members are introduced and then the algorithm can "explore more expansively". When the the mutation probability is very low, the population might "stuck in local optimal", according to Mitchell, M. (1998). In Figure 5, we can see:

- Large mutation probability is preferred when problem size gets larger but not always.
- It's evident that maintaining the mutation probability within a reasonable range is crucial. Deviating from this range necessitates additional function evaluations or duration. However, the impact on extra function evaluations or clock time is relatively minor compared to the optimal mutation probability.
- For consistency, I will employ a mutation probability of 0.2 in subsequent studies.

2) Population Size: Population Size determines how many candidates are selected initially. When population size is large, a large exploration is going on. When population size is small, algorithm can easily stuck at local optimal or take very long time to find the optimal.

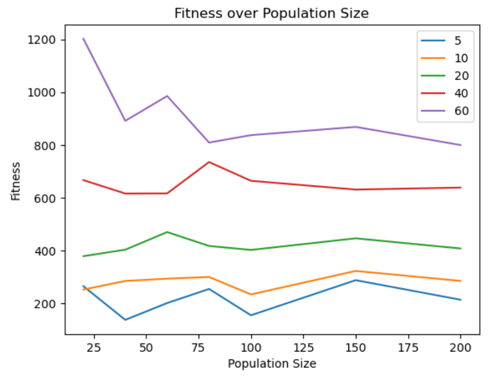

Fig. 6. Fitness over Population Size in TSP

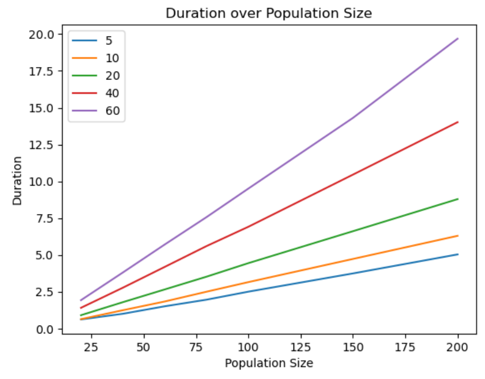

Fig. 7. Duration over Population Size in TSP

Typically, larger problem sizes necessitate larger population sizes. As illustrated in Figure 7, an increase in population size correlates with a corresponding increase in duration. As depicted in Figure 6, for instance, when the problem size is 60, fitness scores tend to rise significantly with population sizes below 20, only to decrease as the population size exceeds 20 and approaches 100.

Subsequently, population sizes will be adjusted based on the best fitness score attained during optimization.

B. *Performance of various algorithms*

- As iterations increase, the performance of all three algorithms generally improves. However, when iterations are held constant, Genetic Algorithm tends to achieve a lower fitness score, indicating a better performance compared to the other algorithms.

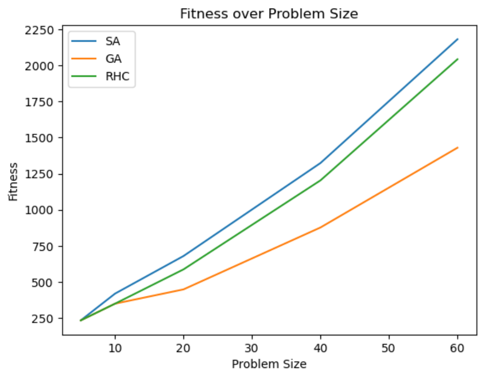

Fig. 8. Fitness over Problem Size in TSP

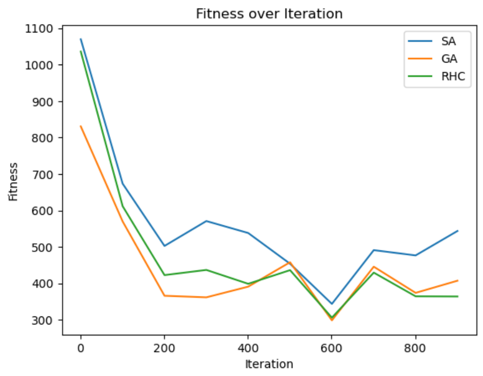

Fig. 9. Fitness over Iterations in TSP

- As the problem size increases, Genetic Algorithm consistently exhibits a lower fitness score compared to the other two algorithms, as shown in Figure 8. Notably, as the problem size escalates, this disparity in fitness scores between Genetic Algorithm and the other two algorithms widens further. This verifies my hypothesis that "heuristic and metaheuristic algorithms including Genetic Algorithm can effectively tackle intricate problems like the TSP.
- For the same number of iterations, Genetic Algorithm consistently outperforms the other two algorithms, particularly when the iteration count is below 600, as illustrated in Figure 9. This superiority stems from Genetic Algorithm's ability to initiate optimization from multiple candidate solutions and employ crossover operations, enabling it to gather more diverse information and refine solutions more effectively within each iteration.

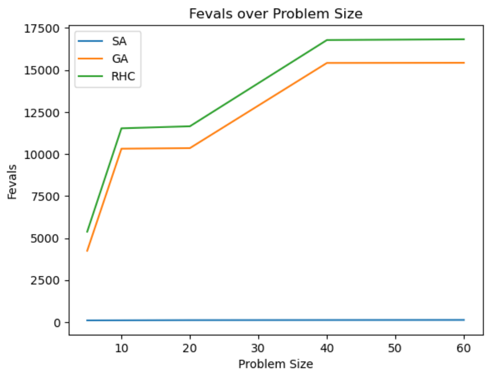

Fig. 10. Fevals over Problem Size in TSP

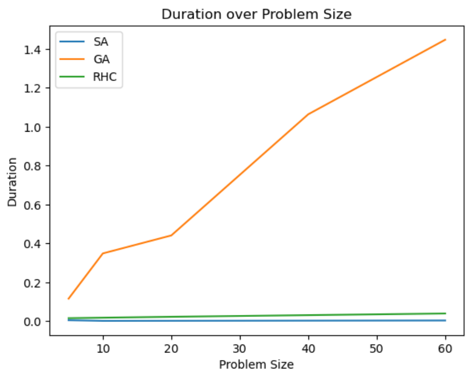

Fig. 11. Duration over Problem Size in TSP

- In exchange for superior performance, Genetic Algorithm typically requires significantly more time to converge and more iterations compared to the other two algorithms. This is clearly shown in Figure 10 and 11. As mentioned above in N-Queens section, this is due to Genetic Algorithm starts from large population size of candidates instead of one or two candidates. Moreover, during the crossover stage, more iterations are needed.

C. *Best Algorithm to solve Travels Man Problem – Genetic Algorithm*

Indeed, Genetic Algorithm emerges as the most effective algorithm for this problem primarily due to its ability to achieve the lowest fitness scores and optimal performance, especially when its hyperparameters are meticulously fine-tuned. However, it's essential to acknowledge that Genetic Algorithm's exceptional performance comes at the cost of significant computational resources and time investment.

IV. NEURAL NETWORK

A. *Dataset*

In Assignment2, I will use the same hospital dataset I used in A1. This dataset is an electronic Health Record prediction collected from a private hospital in Indonesia. It includes critical laboratory metrics like hematocrit and hemoglobin levels and classifies patients into two categories: currently receiving care and completed treatment. Unlike the Iris dataset, this dataset is much larger, with over 4000 data points. However, only two features have a correlation with the target variable above 0.2.

B. Accuracy Score Selection

In this study, we will continue to use the F1 score as the performance measurement metric. As indicated in A1, the F1 score provides a balanced measure for unbalanced datasets. Our hospital dataset is a classic example of an imbalance, where the counts of 1 and 0 in Y are very different — there are more healthy data points than unhealthy data points as shown in figure 12.

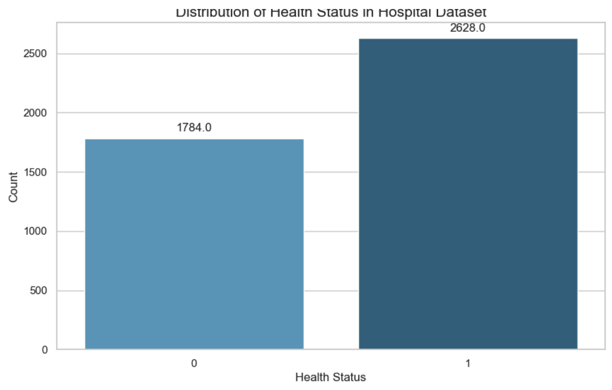

Fig. 12. Unbalanced Dataset

C. Cross Validation

Like A1, in this study, we will use cv = 5 for all future studies. Does CV help to improve the overall performance? Yes, it helps in improving accuracy score a little bit as shown in the table below. This is related to the fact that our hospital data is very unbalanced. However, it's significantly time-consuming considering that one Genatic Algorithm run already takes such a long time.

Algorithm	Testing Accuracy	Testing Accuracy with CV
GA	0.682	0.66
SA	0.64	0.63
RHC	0.64	0.63
GD	0.7	0.67

TABLE II
COMPARISON OF DIFFERENT DECAY RATE IN N-QUEENS PROBLEM

D. Hyperparameters

1) Learning Rate: In neural networks, the learning rate controls the size of the step taken during optimization when updating the model's parameters (weights and biases) to minimize the loss function as referenced from Ruder, S. (2016).

As shown in figure 13, for all optimization algorithms, when learning rate equals to 0.001, the F1 score reaches the highest so we will use it in later studies.

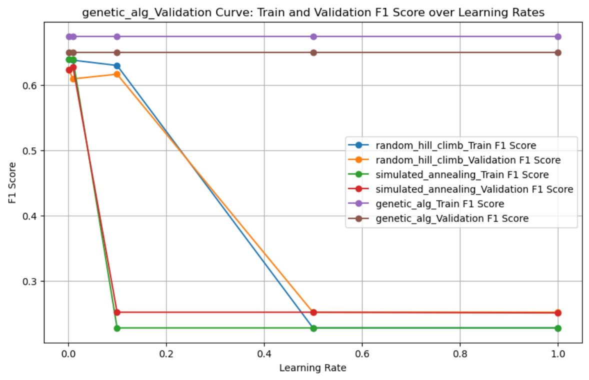

Fig. 13. Validation Curve of Learning Rate

It is worth noting that when the learning rate exceeds 0.1, RHC experiences a significant decrease in F1 score. Similarly, when the learning rate exceeds 0.01, SA also shows a significant drop in F1 score. This phenomenon occurs because high learning rates can cause the algorithms to make too large steps during optimization, which jumps over the optimal solution and results in poor performance. For GA, the learning rate size does not affect F1 score. Why is that? It might because GA, instead of using learning rates to update solutions, relies on selection, crossover, and mutation processes to evolve the population of solutions over generations. This evolutionary approach does not involve gradient descent steps, so the concept of a learning rate is not applicable.

2) Hidden Layer Size: I tried different hidden layers, 5, 10, 15 and 20.

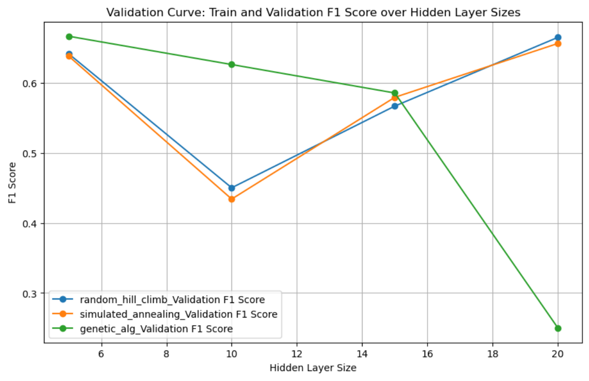

Fig. 14. Validation Curve of Counts of Hidden Nodes

We observe that GA's F1 score decreases as the hidden layer size increases. One key reason for this could be overfitting; when the model overfits, its testing accuracy decreases. Increased model complexity is another factor. A more complex model without additional parameter tuning can perform poorly. Increasing the "max-iteration" to 1000 might improve the accuracy results.

For SA and RHC, the F1 score starts high with a hidden layer size of 5, then decreases, and later increases back to the initial level. With a small hidden layer size, the model can capture essential patterns while avoiding overfitting. As the hidden layer size increases, the likelihood of overfitting also increases, leading to a decrease in the F1 score. When the hidden layer size exceeds 10, the model captures more patterns, resulting in an improved F1 score.

Figure 14 shows that the F1 score is highest when the hidden layer size is 5 for all optimization algorithms. Therefore, we will use a hidden layer size of 5 in subsequent studies.

E. Learning Curve

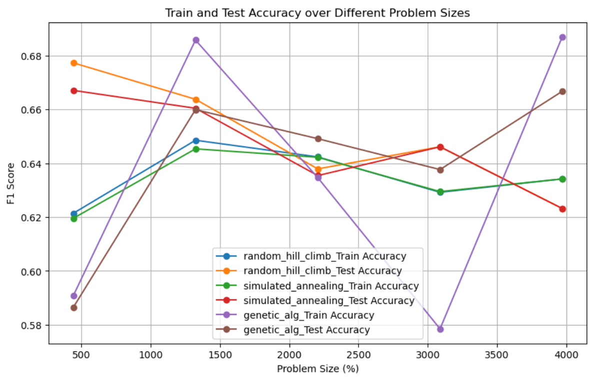

Fig. 15. Learning Curve of NN of Hospital Dataset

Before exploring new optimization algorithms, let's revisit the performance of gradient descent in A1. The optimal neural network model for the hospital data achieves a testing accuracy of 0.67. This model comprises a single layer with 10 neurons and a learning rate of 0.01. The testing duration is 0.7 seconds, while the training duration is 0.73 seconds. Now let us review performance of the other 3 algorithms:

- Among the four optimization algorithms tested, Genetic Algorithm demonstrates significantly longer execution times compared to Simulated Annealing, Gradient Descent, and Randomized Hill Climbing. Despite differences in execution time, all four algorithms achieve similar accuracy levels, approximately 0.65. The ranking based on accuracy from highest to lowest is Genetic Algorithm ¿ Simulated Annealing ¿ Gradient Descent ¿ Random Hill Climbing.
- Simulated Annealing outperforms Randomized Hill Climbing on the hospital dataset due to its ability to handle the complexity of the problem and the large dataset size. Random Hill Climbing, being a simple optimization algorithm, is prone to getting stuck in local optima, limiting its effectiveness on such challenging problems.
- The instability in Genetic Algorithm's performance can be attributed to the limitation of a low maximum iteration setting, such as 100. However, upon increasing the maximum iteration to 1000, Genetic Algorithm consistently demonstrates the best performance among the four algorithms. This improvement highlights Genetic Algorithm's capacity to effectively analyze intricate problems and conduct global search, enabling it to achieve superior results compared to other algorithms.Besides, I didn't do hyperparameter tuning for each problem size in Neural Network problem. I only selected learning rate and hidden layer size based on the largest problem size. This might introduce the instability of GA's performance.
- There are several areas where improvements can be made. For instance, hyperparameter tuning for each problem size in the Neural Network problem could stabilize GA's performance. Additionally, experimenting with more hidden layer configurations, such as (5,5) or (5,5,5) instead of just (5,), might enhance the performance of algorithms like GA and GD. Furthermore, increasing the "max-iteration" parameter to 1000 could potentially yield better results. However, this option was not pursued primarily because GA already requires a significant amount of time to run.

F. Time Duration

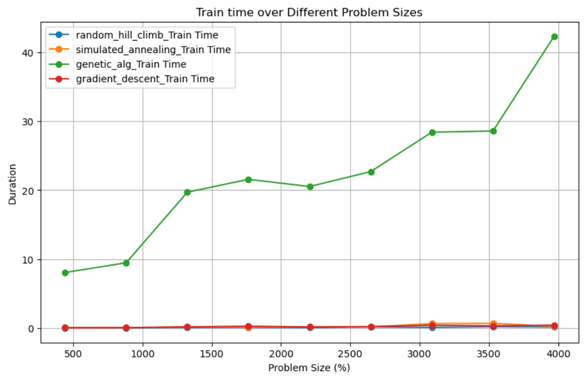

Fig. 16. Train Time of NN of Hospital Dataset

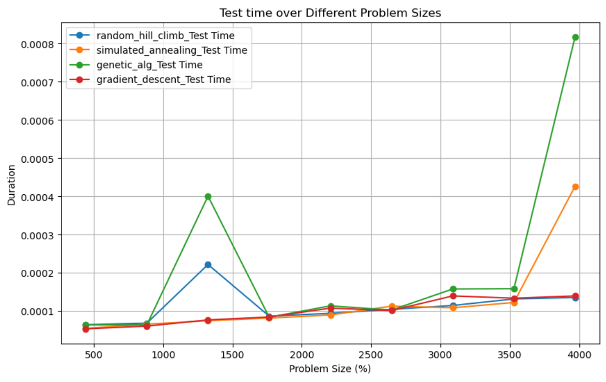

Fig. 17. Test Time of NN of Hospital Dataset

- As shown in figure 16, the training time for Genetic Algorithm is notably longer compared to the other three algorithms, while the training times for the remaining three algorithms are similar. When problem size increases, training time of Genetic Algorithm also increases exponentially. This difference can be attributed to Genetic

Algorithm's population-based approach, where it operates on multiple candidate solutions concurrently, involving time-consuming processes such as crossover operations. When problem size increases, possible combinations also increase exponentially.

- Regarding testing time, in figure 17, all four algorithms are not significantly different from each other. Genetic Algorithm takes a bit longer time but not much.

G. Loss Curve

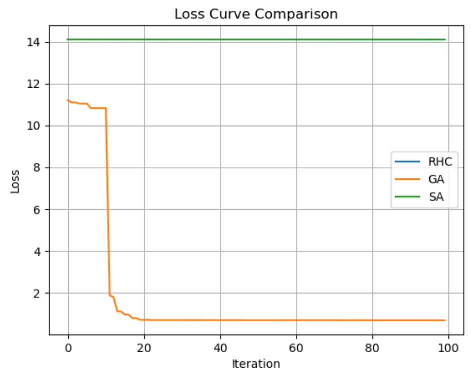

Fig. 18. Loss Curve of NN of Hospital Dataset

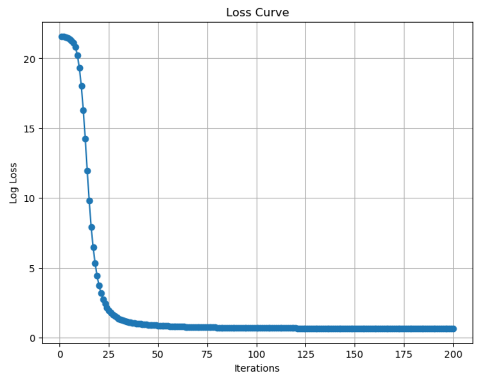

Fig. 19. Loss Curve of Gradient Descent of NN of Hospital Dataset

- Simulated Annealing's Smoothness: Despite not being immediately apparent from figure 18, a closer examination of Simulated Annealing's loss curve reveals a gradual decrease from 14.1 to 14.0. This suggests a slow annealing process due to the initially set slow cooling speed.
- Random Hill Climbing's Smoothness: From figure 18, we can see that Random Hill Climbing, depicted by the overlapping curve with Simulated Annealing, exhibits a similar smoothness. This may indicate rapid convergence or premature trapping in local minima.
- Genetic Algorithm's Decreasing Pattern: From figure 18, we can see Genetic Algorithm's loss curve displays a discernible decreasing pattern, suggesting that the crossover operation leads to the discovery of more optimal candidates and subsequently lower loss. Convergence is typically observed around 20 iterations.
- Gradient Descent's Decreasing Pattern: From figure 19, we ca see similarly, Gradient Descent's loss curve exhibits a decreasing pattern, albeit slightly slower than Genetic Algorithm. Convergence also occurs around 20 iterations, indicating a gradual but consistent improvement in optimization.

H. Best Performance

In summary, I consider Genetic Algorithm to be the top-performing model primarily due to its superior testing accuracy. Despite its longer runtime compared to other models, its quicker convergence allows us to set a lower maximum iteration limit in practice, making it a practical choice for many applications.

V. Summary

- Simulated Annealing has indeed outperformed both Genetic Algorithm and Random Hill Climbing in solving N-Queens problems. This hypothesis was confirmed as Random Hill Climbing, with its tendency to prematurely converge to local minima, struggled in scenarios where even minor adjustments to the current state resulted in conflicting queens. Additionally, Simulated Annealing's simpler management of the cooling schedule contributed to its superior performance compared to Genetic Algorithm, which may require more iterations and computational time.
- Genetic Algorithm has demonstrated excellence in solving Traveling Salesman Problems (TSP), especially as the problems became more complex with increasing problem size. This aligns with the inherent ability of heuristic and metaheuristic algorithms, including Genetic Algorithm, to effectively tackle intricate problems like the TSP. However, the train time duration is extremely long, so does the count of fevals, all related to its internal complicated algorithm.

References

[1] Gutin, G., & Punnen, A. P. (Eds.). (2007). *The Traveling Salesman Problem and Its Variations.* Springer Science & Business Media.

[2] Bell, C., & Stevens, S. (2009). *A Survey of Known Results and Research Areas for N-Queens.* Discrete Mathematics, 309(1), 1-31.

[3] Bäck, T., Fogel, D. B., & Michalewicz, Z. (1997). *Handbook of Evolutionary Computation.* Oxford University Press.

[4] Mitchell, M. (1998). An Introduction to Genetic Algorithms. MIT Press.

[5] Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747. Link