# CS7641 A2: Randomized Optimization

Charles Leung
*cleung75@gatech.edu*

Optimization is a common tool used to solve many problems we have in everyday life. Whether this may be a cost function we may be trying to minimize for a company's budgeting system, or maximizing points for a specific system, they all can be integrated with randomized optimization tactics. When using such optimization tactics, however, it is key to understand the behaviors of each optimization algorithm and select the appropriate ones (with its associated hyperparameters), to generate the best results. In this report, we will examine two common optimization problems, and compare and contrast the performance of three different optimization algorithms with such problems: Random Hill Climbing (RHC), Simulated Annealing (SA), and Genetic Alg (GA). The problems I have chosen are N-Queens, and Continuous Peaks. In addition, we will also revisit the rice classification dataset from UCI, and use a neural network based learner and compare its original performance with RHC, SA, and GA as its primary optimization algorithm.

RHC utilizes random solutions and iteratively makes incremental changes (at random) to converge to a solution. It will only accept that new solution if it provides better fitness scores. This is a simplistic and easy algorithm to implement but can easily get stuck in local optima. SA, analogous to an annealing process in blacksmithing, utilizes a temperature parameter that reheats/cools to encourage further exploration and avoid being stuck at local optima. GA handles problem sets by utilizing population density and combining two "genetic" results of the parent solutions to produce children that may/may not have better fitness. It also utilizes mutation values which can have randomized traits in its offspring. These two features can allow for diversification for a greater probability to find the true optimal solution.

N-Queens and Continuous Peaks were chosen as they have different fitness functions and should perform and behave differently. To begin, N-Queens is a combinatorial optimization problem, which focuses on minimizing the number of queens in an N x N chess board that can attack one another. This is also a very notoriously difficult algorithms problem in which I have spent many nights trying to solve and understand the mathematics behind this problem, therefore, it would be interesting to see how the optimizer tackles such an algorithm. On the other hand, Continuous Peaks is a binary optimization problem, which will produce a different analysis result when optimizing against different algorithm structures. The difference in the two problems will generate more interesting conversations when comparing them together. In our example, we will take advantage of the fitness and algorithm api functions from mlrose_hiive.

The second part of the experiment will extract a real world dataset given to us from the UCI Machine Learning repository, which classifies rice into two different types, Cammeo and Osmancik. We will use a neural network learner, given to us from mlrose_hiive, and generate the most optimal algorithm to train the learners on. We can then explore the differences and analyze the learners performance.

Given the two optimization problems, we think that GA is the most robust and can provide the best convergence to maximize fitness. However, the trade off is that the runtime of GA is impacted due to its complexity. Because of the complexity of N-Queens (very restrictive rules in its fitness function), we think SA may be best optimized for N-Queens and GA for continuous peaks.

## I. Optimizer/Algorithm Explanation

### A. N-Queens

As stated before, N Queens is a combinatorial optimization problem that aims to optimize the number of queens that can live on an N x N board without any two queens attacking one another. In chess, the queen is able to move (and attack) any amount of squares that share the same row, column, or diagonal as them. The essence of this problem now is to evaluate the state (fitness) of a current board set up that maximizes the amount of queens that can co-exist in on a board of size N x N. When boiling this down, this means we must select a valid pair from N queens, or N choose 2. This combinatorics problem reduces to the simple equation below.

$$\binom{N}{2} = \frac{N(N-1)}{2}$$

This represents the amount of ways to choose two items of unique pairs of queens without interference. Therefore, when representing NC2, the maximum fitness is illustrated in the equation below. One noteworthy comment about N Queens is that the problem landscape grows exponentially as N increases by $N^N$. What this means is that the optimizer will have an exponentially increased area to search from as the number of possibilities grow, which will heavily increase the amount of computation that the algorithms will undergo and provide a good analysis on the different optimizers.

When we implement the algorithms and optimize, we will use this maximum function to quantitatively analyze its fitness performance.

### B. Continuous Peaks

Continuous peaks aims to maximize the length of continuous sequences of 0s and 1 in a binary string. An additional threshold parameter, T is the minimum length it must exceed to score a bonus to the fitness function. The final fitness function is then characterized below, where x represents the length of the binary string and T being the threshold.

$$\text{Fitness}(x,T) = \max(\text{max\_run}(0,x), \text{max\_run}(1,x)) + \begin{cases} n & \text{if max\_run}(0,x) > T \text{ and max\_run}(1,x) > T \\ 0 & \text{otherwise} \end{cases}$$

The maximum fitness that can be obtained in any scenario is 2n. If we were to closely examine this more carefully, the maximum length of the longest continuous sequence of 0s and s is n. The reward component adds n to the fitness score when the longest runs of both 0s and 1s exceed threshold. This makes the final score n + n, or 2n. This can be used in our analysis when comparing the algorithms against one another. Combination possibilities also increase exponentially, but in base 2 instead of n now, since every entry has to be a 0 or a 1.

## II. Random Hill Climbing (RHC)

### A. N-Queens

Random Hill Climbing (RHC) is the simplest optimization algorithm used to solve various applicable combinatorial/continuous problems. At a high level, RHC's goal is to converge to a maxima in a function by iteratively improving a single solution, moving inch by inch incrementally based on the local changes observed by its neighbors. The optimizer will randomly make local changes and accept solutions if they provide better fitness. One drawback of RHC however, is that it is prone to converging to local minimum instead of the global optimum. Mlrose's repository utilizes restart values periodically to inject diversification/encourages the searcher to explore the landscape more often.

We build out the best RHC algorithm by iteratively performing a grid search to loop through the different hyperparameters that are consumed in the RHC algorithm. Beginning with N-Queens, the hyperparameters we will be investigating (including the ranges) we will be exploring are number of restart opportunities and max attempts.

To have a better understanding of each hyperparameter and determine its effects on our overall fitness, we plot the fitness trend with all constant parameters except for the investigating hyperparameter, which alters its value based on the list above. Figure 1a-d shows these results.

Beginning with figure 1a, the goal of the restart allowance is to help the algorithm periodically restart the search with a new random initial solution. This is

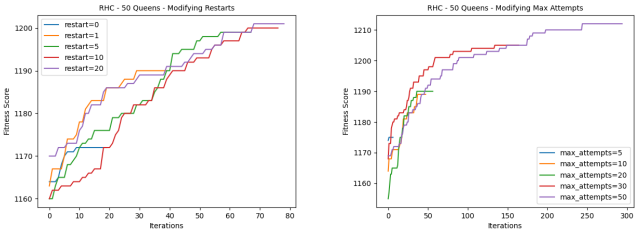


Fig. 1: 50-Queens Hyperparameters. From left to right: (a) RHC algorithm fitness vs iterations, varying restart allowance, (b) RHC algorithm fitness vs iterations, varying max attempts

paramount in RHC, as it relies on making actions based on the neighbor's fitness landscape. Because of this, it is prone to bias and ultimately may converge to a local optima. By restarting at different landscapes, the algorithm is allowed to explore different areas, and the best solution across all restarts is returned as the final result. From this information, it is obvious that we want to have a fair number of restart parameters as we want to explore the problem space as robust as possible. Therefore, as accurately displayed in figure 1a, we can see that our theory is correct, with restarts=20 providing the highest fitness score, nicely converging to the maximum fitness of 1225 (since figure 1a is using 50 queens). It is important to note that we want just enough restarts to provide a good convergence to the global optima, as having an overly generous restart parameter will be quite overkill and exhaust computational resources.

Maximum fitness is the number of iterations the algorithm will perform without finding an improvement in the fitness value before terminating or restarting. This encourages diversification and prevents local minimum convergences, while also ensuring that the algorithm efficiently uses its computational resources. In figure 1b, we can see that having a high amount of maximum attempts will (naturally) extend the number of iterations since we allow more tolerance to explore, which will result in the algorithm ultimately having better opportunities to provide a higher fitness score. This is indicated with max_attempts = 50 yielding a closely converged score to the max score.

Now that we know the two interesting parameters, we will now go through a grid search and explore the parameter values that will yield us consistently high fitness results. To get a better understanding of the problem size, we have also altered the problem size (the N values in N-Queens) to see how the algorithm behaves in terms of run time and computational efficiency with an increased problem space. The results and the metadata are then displayed in figure 2 and table 1 respectively.

Since we know that there is a factor of randomness for each run because of the initial state location being different every time, we took 5 random seeds and ran each problem size 5 times and took the average to prevent any confirmation bias. This explains the high amounts of

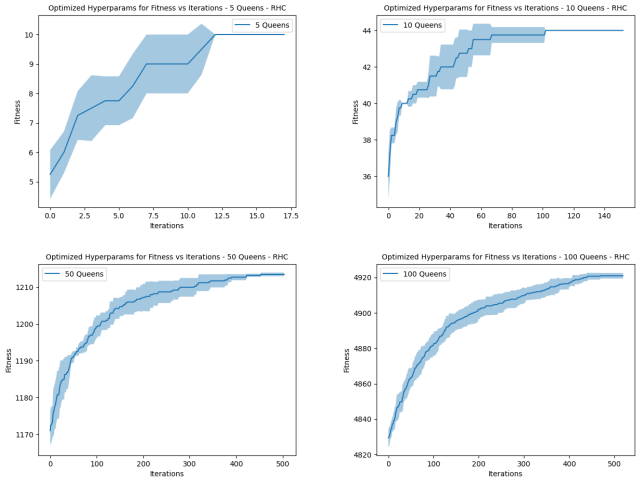


Fig. 2: Fitness curves for RHC grid search, N-Queens. From left to right: (a) 5-Queens, (b) 10-Queens, (c) 50-Queens, (d) 100-Queens

TABLE I: RHC N-Queens Statistics

| N-Queen | Fitness | Max Fit. | Restart | Attempts | Calc Time | FeVals |
|---|---|---|---|---|---|---|
| 5 | 10 | 10 | 5 | 5 | 0.0005 s | 57270 |
| 10 | 44 | 45 | 10 | 50 | 0.023 s | 183279 |
| 50 | 1213 | 1225 | 10 | 50 | 1.23 s | 441450 |
| 100 | 4921 | 4950 | 20 | 50 | 18.92 s | 664928 |

variance as the optimizer continues to journey towards the final solution, where there is an evident reduction in variance overtime and ultimately converges very closely to the final score, as indicated by the statistics in table I. As we expected, a higher restart and attempt value is required as the problem space increases, since the space increases exponentially and the algorithm needs more opportunities to explore. Another important thing to analyze is FeVals, which is the number of ”processes” performed during the optimization process (not to be confused with iterations, which is the number of overall steps). This is due to the exponential growth in search space, and also the need for more restarts to find optimal solutions, which ultimately requires more evaluations.

### *B. Continuous Peaks*

We will repeat the approach for continuous peaks. The hyperparameters we will be investigating we will be exploring are going to be exactly the same as the ones in N-Queens. Due to lighter computation power, I have also increased the range to provide a better analysis, as shown in figure 3.

We will now go through a grid search and explore the parameter values that will yield us consistently high fitness results. As with N-Queens, we have also altered

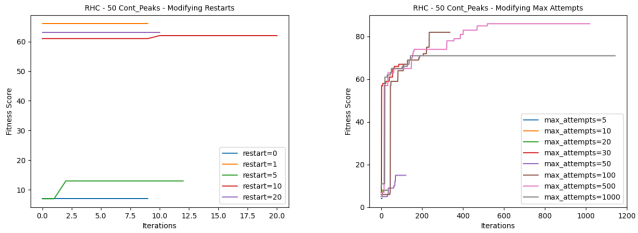


Fig. 3: Cont. Peaks Hyperparameters (String Length = 50). From left to right: (a) RHC algorithm fitness vs iterations, varying restart allowance, (b) RHC algorithm fitness vs iterations, varying max attempts

the problem size (the string length) to see how the algorithm behaves in terms of run time and computational efficiency with an increased problem space. The results and the metadata are then displayed in figure 4 and table II respectively.

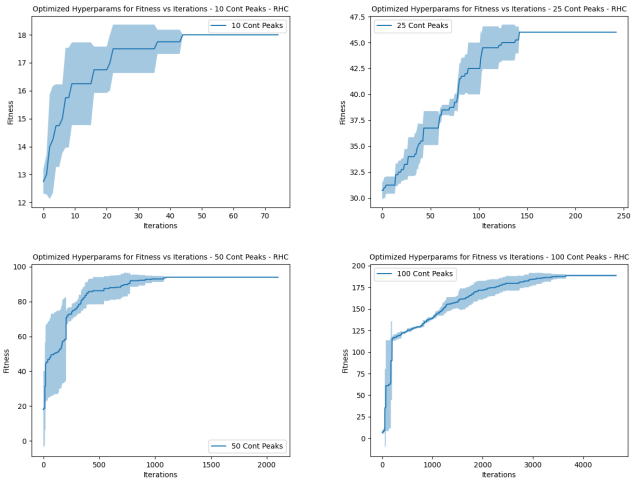


Fig. 4: Fitness curves for RHC grid search, Continuous Peaks. From left to right: (a) 5-Continuous Peaks, (b) 25-Continuous Peaks, (c) 50-Continuous Peaks, (d) 100-Continuous Peaks

TABLE II: RHC N-Continuous Peaks Statistics

| Cont Peaks | Fitness | Max Fit. | Restart | Attempts | Calc Time | FeVals |
|---|---|---|---|---|---|---|
| 10 | 18 | 20 | 0 | 30 | 0.0003s | 10098 |
| 25 | 46 | 50 | 5 | 100 | 0.0100s | 51594 |
| 50 | 94 | 100 | 5 | 1000 | 0.088s | 770103 |
| 100 | 188.5 | 200 | 20 | 1000 | 1.304 s | 3062598 |

Like N-Queens, we took 5 random seeds and ran each problem size 5 times and took the average to prevent any confirmation bias. From observation, there are high amounts of variance as the optimizer continues to jour-

ney towards the final solution, where there is an evident reduction in variance overtime and ultimately converges very closely to the final score. What is interesting, though is that there there is a reduction in accuracy in some of the runs, for example, with 10 Queens and 10 continuous peaks, the fitness converged less optimally (44/45 vs 46/50). This suggests the validation of our theory, that the rugged of the problem (with many local optima) are less ideal for RHC, which heavily relies on its neighbors. Another notable feature is the FeVals being much lower than Queens (almost half as much evaluations). This is because of the complexity of the problem and the problem size being much smaller $2^N$ in Cont Peaks in comparison to $N^N$ in Queens, and with less fitness landscape, there are less evaluations to pursue. From this, the clock time also substantially reduces from the reduction in evaluations necessary.

## III. Simulated Annealing (SA)

Simulated annealing is derived from the annealing process performed by blacksmiths and scientists. The repeated heating and cooling of metals allow the material to soften the metal and reduce any internal stresses and increases the ductility of the metal. As the theory suggests, we will too be alternating this theory by "cooling" the problem from the initial temperature, and introducing a probability factor of it conditionally accepting a worse solution, to introduce more diversity. The approach here will be similar to the RHC - we will be first examining the hyperparameters, and then performing grid search to find the optimal fitness score based on 5 randomized instances. The hyperparameters and the ranges we will be exploring are initial temperature, max attempts, and initial starting temperature.

### A. N-Queens

With the approach discussed, we can dive right into exploring the three hyperparameters. The results are indicated in figure 5.

Beginning with Fig 5a, decay schedules seem to be consistently providing the same fitness evaluation. However, it is notable that arithmetic decay takes more to converge and explores a lot more of the landscape before completing the convergence. This is because the temperature decreases linearly with every iteration and slows down the convergence. This is good for problems with many local optima, but is much more computationally expensive. Exponential and Geometric decay provides a much faster convergence by uses exponential and multiplicative temperature decay schedules respectively. The trade off with this is that there is a higher risk of it reaching a local optima.

Initial temperature (Fig 5b) showcases the effectiveness of the initial temperature to begin the simulation. The temperature parameter determines how freely the algorithm can explore the search space by accepting worse solutions, since the probability factor is determined with temperature as a factor. The higher the temperature, the more erratic the fitness journey. Interestingly, it is shown here that a low temperature generated a higher fitness score. This is interesting, as it may suggest that the initial region of the search space may already be near an optimal solution, and less acceptance of deviation will hone into the optimization faster. It is also important to note that we are using only one instance. Therefore, when we complete the grid search, it will be done through multiple seeds so we can eliminate potential false positives.

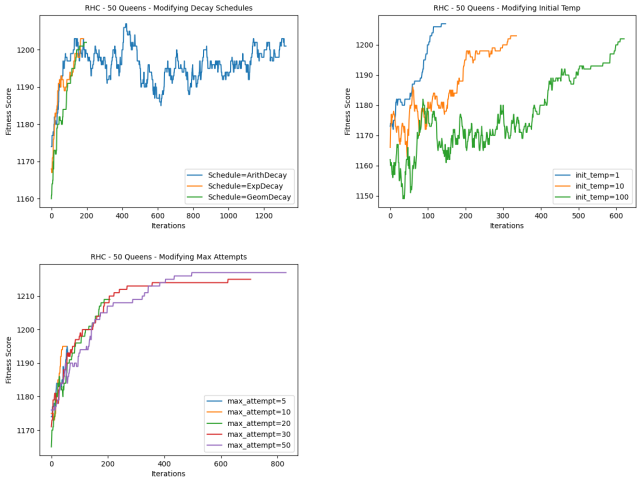


Fig. 5: 50-Queens SA Hyperparameters. From top left to bottom right: (a) Decay Schedule, (b) Initial Temperature, (c) Max Attempts

Max attempts (Fig 5c) behave similarly to RHC - it caps how many attempts we are allowed to continue on before terminating. We can see a clear display of diminishing returns, as when max = 20 and onwards provided consistently high fitness values.

With this in mind, we begin our grid search and the results are displayed below in Figure 6, with the metadata listed in table III. The problem size will be kept consistent with the values given in RHC.

TABLE III: SA N-Queens Statistics

| N-Queen | Fitness | Max Fit. | Sched. | Max Att. | Init. Temp | Calc Time | FeVals |
|---|---|---|---|---|---|---|---|
| 5 | 10 | 10 | Exp. | 50 | 100 | 0.014s | 1200 |
| 10 | 44.5 | 45 | Arith | 50 | 1 | 0.159s | 4364 |
| 50 | 1215.75 | 1225 | Exp | 50 | 10 | 0.729s | 1171 |
| 100 | 4933.5 | 4950 | Exp | 50 | 100 | 7.518s | 2389 |

The results here are quite interesting. Beginning with the results, we can see there is some convergence with all of the problem sizes. As expected, runs with high initial temperatures (5 and 100 Queens) tend to have more erratic behavior near the beginning, as higher temperatures behave closer to a random walk than lower initial

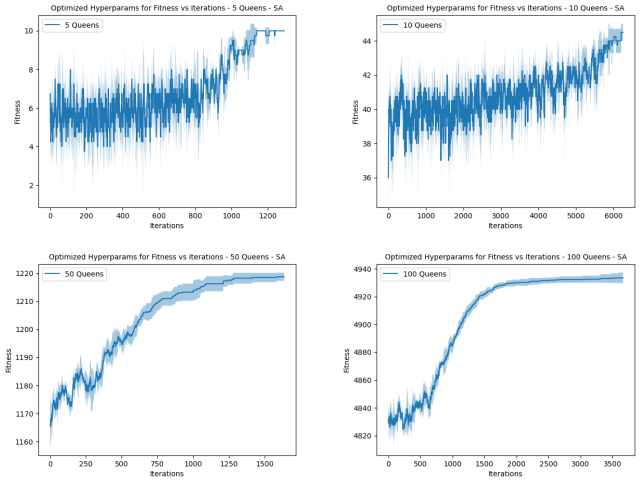


Fig. 6: Fitness curves for SA grid search, N-Queens. From left to right: (a) 5-Queens, (b) 10-Queens, (c) 50-Queens, (d) 100-Queens

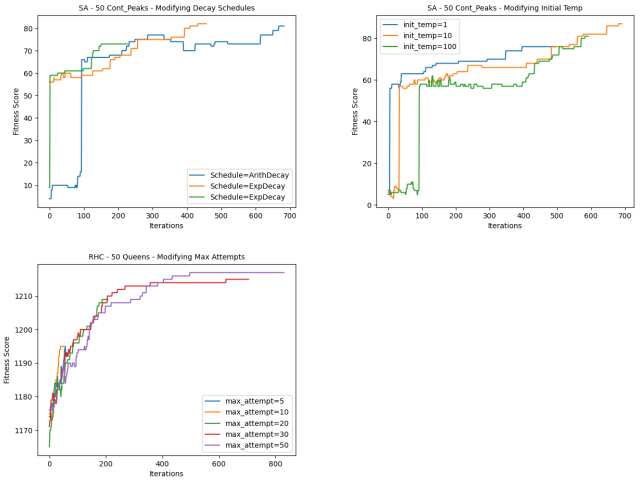


Fig. 7: 50-Continuous Peaks SA Hyperparameters. From top left to bottom right: (a) Decay Schedule, (b) Initial Temperature, (c) Max Attempts

temperatures. Notably with the 10-Queens experiment, we can see a significant increase of iterations due to the schedule chosen. Because of the linearity in decay, we can also see the significant spike in processes in Fitness evaluations in comparison to the other problem sizes. The key takeaway here is that schedule decisions heavily impact exploration, computational efficiency and convergence behavior. Overall, the algorithm performs very well, providing close to optimal scores consistently for every problem size increase. In terms of time efficiency, we can really see the effectiveness of an exp. decay schedule calculating much faster times than what was offered in RHC, with fitness values being more than 277x smaller than RHC, while providing improvement on fitness!

### *B. Continuous Peaks*

We will now repeat the entire experiment cycle with continuous peaks. Figure 7 below displays the hyperparameters we will be experimenting with.

Observations are consistent with those displayed in N-Queens. As expected, the temperature and decay schedule heavily impacts the time taken to reach convergence. However, judging from the iteration we have now, it looks like a simpler schedule can also perform extremely well (like Geom/Exp. decay instead of Arithmetic). Observations for max attempts also match with those from N-Queens, having more opportunities to explore (higher attempt) will yield a greater fitness.

We now take the parameters and throw it into the grid search to find the most optimal parameters to maximize fitness. Like the other experiments, we also ran this under 5 seeds to remove any bias and properly provide a better convergence. The results and the metadata are displayed in figure 8 and table IV respectively.

The results here again shows nice convergence towards an optimal solution at a fraction of the cost in

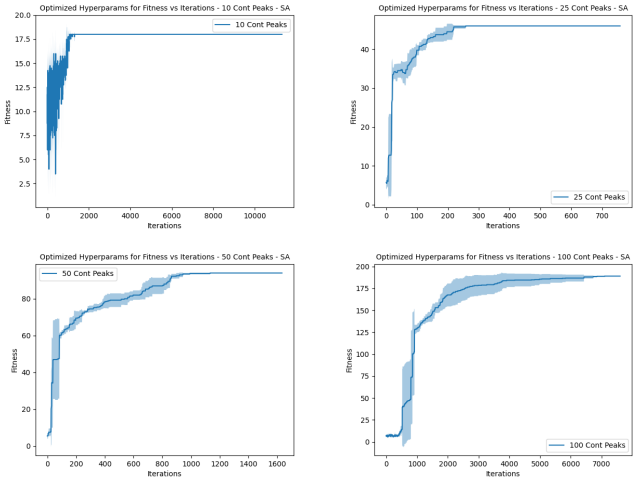


Fig. 8: Fitness curves for SA grid search, Continuous Peaks. From left to right: (a) 10-Cont Peaks, (b) 25-Cont Peaks, (c) 50-Cont Peaks, (d) 100-Cont Peaks

compute power. We can also see the tradeoffs between different hyperparameters. For example, at 10 peaks, the high initial temperature causes a lot of variance when travelling towards the optimal, as seen in the massive variance before convergence. When convergence happens, it tapers off nicely towards a conclusive solution. This convergence is also apparent on the otehr runs as we increase the problem size. Here we can see how Exponential Decay displays its quickness in convergence in comparison to a geometric convergence as well, given the little iterations needed to decay in a geometric setting. The big win here however, is how well and performant it can be in comparison to RHC. In RHC, we can see that there is much more function evaluations and calculation time to convergence, for the same if not worse result given from SA. This proves that SA is more suited for a case like this, perhaps due to the rugged

TABLE IV: SA Continuous Peaks Statistics

| Cont Peaks | Fitness | Max Fit. | Sched. | Max Att. | Init. Temp | Calc Time | FeVals |
|---|---|---|---|---|---|---|---|
| 10 | 18 | 20 | Exp. | 10000 | 100 | 0.09s | 6431 |
| 25 | 46 | 50 | Geom | 500 | 1 | 0.007s | 444 |
| 50 | 94 | 100 | Exp | 500 | 1 | 0.187s | 1075 |
| 100 | 189 | 200 | Geom | 500 | 100 | 0.084s | 4542 |

nature of the problem's landscape.

## IV. Genetic Algorithms (GA)

Inspired by natural selection and genetics, genetic algorithms (GA) maintains a population of potential solutions that drift overtime crossing over genes, mutate, and ultimately create superior offspring with increased fitness. The hope is that the repeated genetic reshuffling and mating will allow the algorithm to search through the landscape and ultimately find an optimal solution. The analysis techniques here will also be repeated from SA and RHC: first analyzing hyperparameters, then tuning, then discussing. The hyperparameters we will be analyzing are population, mutation rate, and max attempt.

### A. N-Queens

With the hyperparameters introduced, we begin to dive deep on the affects of each hyperparameter on the algorithm, indicated by Figure 9.

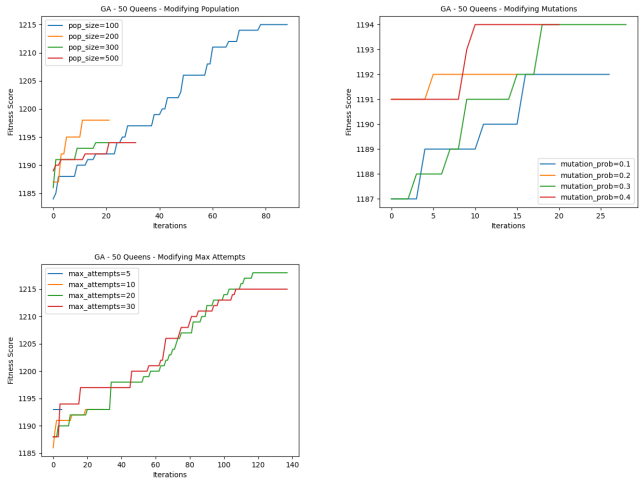


Fig. 9: 50-Queens GA Hyperparameters. From top left to bottom right: (a) Population, (b) Mutation, (c) Max Attempts

Population (Fig 9a) refers to the set of potential solutions to the optimization. The individual in the population represents as a solution, to be crossed over to another. A large population will provide better exploration of the search space and reduce local optima convergence, but will take more time to converge due to the diversity. The opposite is true about a small population. In our graph, we can see that the low population size quickly converges in a short amount of iterations. Therefore, while the plot initially suggests that a pop size of 100 gives good fitness, the large populations showcase their robustness and continues to develop a more accurate high score.

Mutation (Fig 9b) controls the likelihood of a mutation occurring when crossing over genes. This factor helps build randomness into the search and prevents from converging into local optima. The higher the mutation rate, the more random it is, and causes the process to converge faster. In our example, we can see that there is a threshold amount of mutation before they all converge quite nicely. Having a slightly larger mutation will allow the optimizer to converge close to max while a lower mutation will be more diligent in searching, but require more computation.

The parameters are then resimulated to grid search for the most optimal fitness score, shown in figure 10 and table V below.

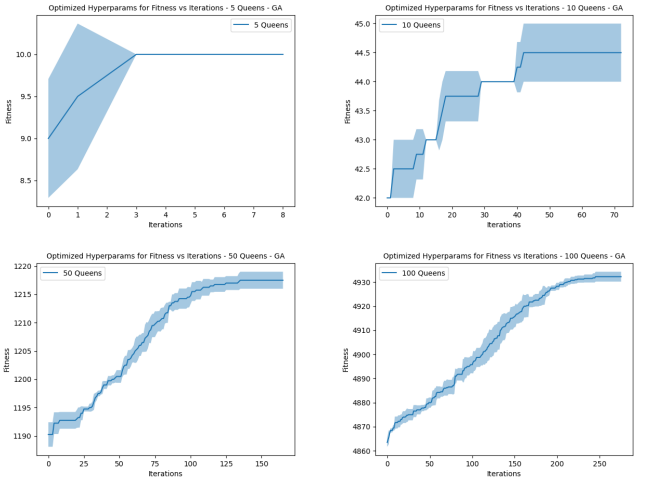


Fig. 10: Fitness curves for GA grid search, N-Queens. From left to right: (a) 5-Queens, (b) 10-Queens, (c) 50-Queens, (d) 100-Queens

TABLE V: GA N-Queens Statistics

| N-Queen | Fitness | Max Fit. | Pop | Max Att. | Mut. | Calc Time | FeVals |
|---|---|---|---|---|---|---|---|
| 5 | 10 | 10 | 100 | 5 | 0.4 | 0.012s | 573 |
| 10 | 44.5 | 45 | 500 | 30 | 0.1 | 0.952s | 17918 |
| 50 | 1217.5 | 1225 | 100 | 30 | 0.2 | 6.621s | 8457 |
| 100 | 4932.25 | 4950 | 200 | 30 | 0.4 | 103.762s | 27963 |

Overall the results showed nice convergence and high overall optimal values that are close to the maximum.

Key takeaways here are that large populations enhance exploration and improve overall fitness, as indicated by 10-Queens' FeVals, and the higher than normal iterations. Having a large population also promotes more search, indicated by the larger variance in Fig 10b. Mutation rates can also play a significant role in diversifying our optimizer, but comes at a cost with a massive calculation time (almost 7 hours to run in comparison to RHC at 2.6 hours). This suggests the importance of balancing the parameters with computation, and computation with Fitness evaluations.

### *B. Continuous Peaks*

We will now repeat the entire experiment cycle with continuous peaks. Figure 11 below displays the hyperparameters we will be experimenting with.

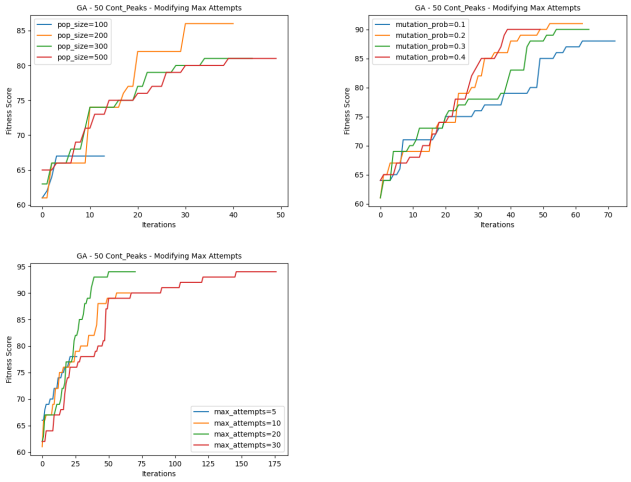


Fig. 11: 50-Cont Peaks GA Hyperparameters. From top left to bottom right: (a) Population, (b) Mutation, (c) Max Attempts

Observations are consistent with those displayed in N-Queens. As expected, the temperature and decay schedule heavily impacts the time taken to reach convergence. We now take the parameters and throw it into the grid search to find the most optimal parameters to maximize fitness. The results and the metadata are displayed in figure 12 and table VI respectively.

TABLE VI: GA N-Continuous Peaks Statistics

| Cont Peaks | Fitness | Max Fit. | Pop | Max Att. | Mut. | Calc Time | FeVals |
|---|---|---|---|---|---|---|---|
| 10 | 18 | 20 | 100 | 5 | 0.2 | 0.011s | 581 |
| 25 | 46 | 50 | 100 | 30 | 0.3 | 0.111s | 3580 |
| 50 | 94 | 100 | 300 | 20 | 0.4 | 0.766s | 23874 |
| 100 | 185.25 | 300 | 30 | 30 | 0.4 | 2.388s | 38300 |

The results here are showing consistent trends of GA's overall robustness. Being able to achieve consistent

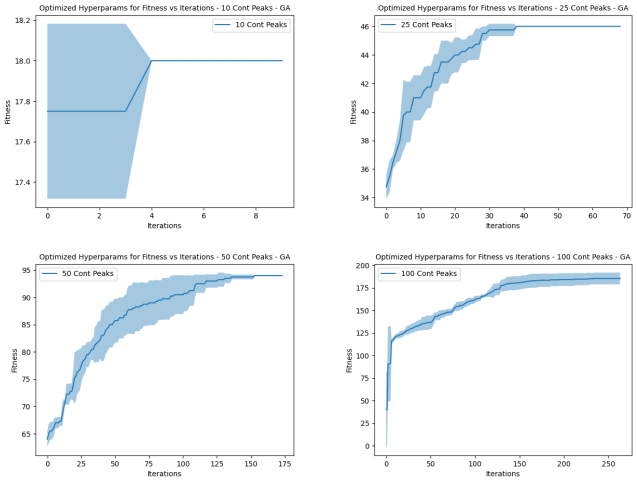


Fig. 12: Fitness curves for GA grid search, Continuous Peaks. From left to right: (a) 10-Cont Peaks, (b) 25-Cont Peaks, (c) 50-Cont Peaks, (d) 100-Cont Peaks

and high fitness values upon tuning population sizes, mutation and max attempts balance exploration and efficiency. All four problem sizes were able to strongly converge into a solution across five different test seeds. What is surprising though is how quick GA can perform here relative to N-Queens. Perhaps due to the complex nature of the N-Queens problem, and the larger search space, the simplicity that comes with Continuous Peaks may make it a much better contender for GA compared to SA. The computation time is slightly slower, but it is traded off with a higher robust method to converge well to an optimal solution.

## V. Neural Networks

The dataset I have used in my previous assignment was the rice dataset. Using Sklearn's MLPClassifier and running GridSearchCV, we were able to determien that the optimal hyper parameters were hidden nodes = 16, learning rate = 0.01, and activation function = 'tanh'. Our approach here will be to rebuild the neural network using MLRose and then apply the 3 algorithms as our weight adjuster. We can then compare our loss curves with the baseline, which is gradient descnt. We used scaling and encoding to replace our X values, and 1 or a 0 to represent our Y values.

The testing data is then split into 5 folds so cross validation can be implemented. We will validate each set with the testing set as well. We first perform grid search on the remaining parameters to find the optimal pairing (ie population in GA). That way we can properly analyze and compare the two algorithms in the closest manner. The results of RHC, SA, GA, and Gradient Descent (back propagation) is displayed in figure 13 below and the metadata on table VII below.

From the results shown, all three algorithms were able to closely match if not exceed the expectations of gradient descent. As mentioned in A1, the dataset was

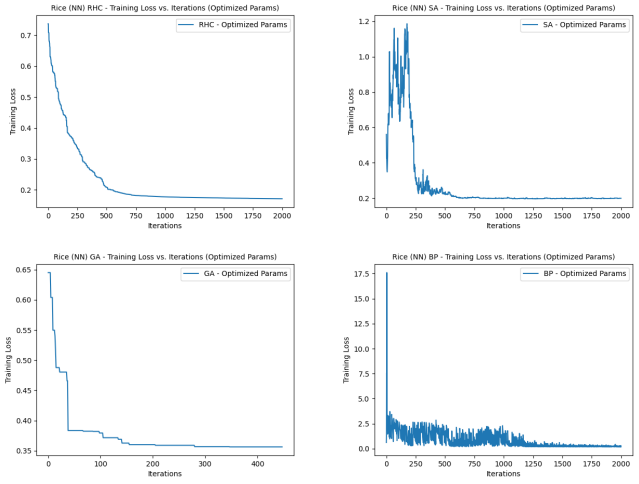


Fig. 13: Rice Dataset - Neural Network Loss Curves. From top left to bottom right: (a) RHC, (b) SA, (c) GA, (d) Gradient Descent

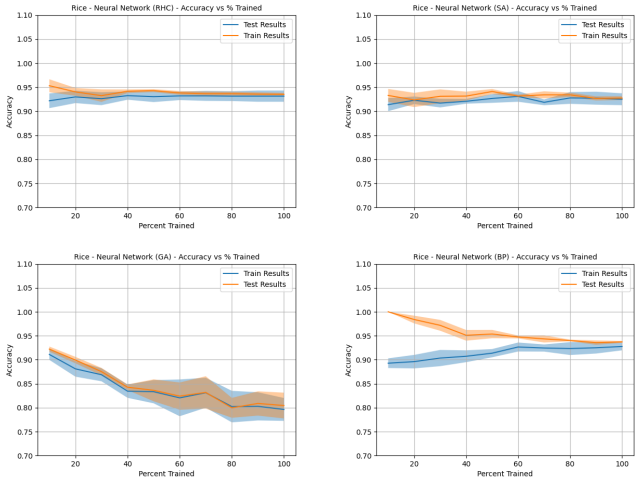


Fig. 14: Rice Dataset - Neural Network Loss Curves. From top left to bottom right: (a) RHC, (b) SA, (c) GA, (d) Gradient Descent

TABLE VII: Random Optimizer Performance Comparison

| Rand. Opt. Algo. | Train Time (s) | Accuracy (%) |
|---|---|---|
| RHC | 51.935 | 92.39% |
| SA | 0.877 | 91.77% |
| GA | 53.174 | 92.30% |
| Grad Descent | 7.719 | 91.60% |

rather easy for the network for it to split and classify. This can be seen from the relatively smooth convergence an optimal solution in all three algorithms. It may also possibly suggest that the complexity of the model is low, with little local minima to provide an easy convergence. One thing we can look for in the future is to also run this with seeds to observe a less bias convergence. However, due to long runtimes of these algorithms and a deficiency in compute power, we avoided that to allow computation for other graphs.

Figure 14 below then indicates the learning curve of all four of the algorithms, which indicates a very nice convergence, which further propels our case that the algorithm was able to handle the data set that was given to us. There is no sign of overfitting or underfitting, meaning it has been sufficiently trained and the fit of the training and testing data is good. Please note that the train and test labels were accidentally reversed in Fig 14b, c, and d. In this case, blue is test data, and orange is train (apologies for the oversight, we were unable to remake the graphs due to the computation being too lengthy).

## VI. Comparison and Conclusion

The results of the experiment were quite interesting for N-Queens and Continuous Peaks. Many notable features were highlighted, such as the optima being closely achieved for both the N-Queens and Continuous Peaks. However, let us now analyze the holistic data across all three algorithms against both problems.

Beginning with wall clock, we can see a common trend across all algorithms that there is a direct, almost exponential relationship between FeVals and calculation time. As the problem size increases, the optimizer needs to consider a lot more states in the landscape and therefore would need to run many more runs per iteration. Hence, we can see how something like 100-Queens in Table I requiring 665k calculations, and a whopping 18.92s to calculate the algorithm. The takeaway is that we must be smart in our parameter choices and optimizer selection when problem size grows - while we increase problem size complexity, the computation efficiency goes down. If one chooses strict parameters that take a while to converge (ie a high initial Temperature in SA), then it will be very computationally expensive to generate an output.

It was obvious that the increased complexity and landscape of N-Queens severely impacted run time and function evaluation processes, particularly in RHC and GA. Where as in continuous peaks, since the problem was less restrained (simpler fitness function, only looking at the length of strings), we can see the drastic decrease in function evaluations throughout and overall a faster wall clock time, as discussed and shown in the tables above. Therefore it makes sense to assign SA to RHC, given its faster computation time and equally as good results, and GA to Continuous Peaks, as the problem size is more manageable and the landscape is better managed from the robustness of GA.