Assignment 2: Randomized Optimization

CS 7641 Machine Learning
Spring 2017

Karel Klein Cardeña

Part 1: Randomized Optimization in Neural Networks

In this experiment, three randomized optimization algorithms, namely Randomized Hill Climbing, a Genetic Algorithm, and Simulated Annealing were used to find weights for a neural network. The performance of these algorithms are then compared to the results of backpropagation found in the supervised learning experiments.

Pima Indians Diabetes Dataset

This dataset was used to test various supervised learning algorithms in assignment 1. It is a subset from the database owned by the National Institute of Diabetes and Kidney Disease and poses an interesting problem. With 768 instances each corresponding to an individual, the data attempts to correlate 8 health metrics with the diagnosis of diabetes. In the included diabetes_*.txt files, a 1 corresponds to a patient that has been diagnosed with diabetes, and a 0 to a patient without the diagnosis.

The early detection of diabetes is crucial in preventing or delaying the onset of serious health consequences. An untreated diabetic will face an increased risk of kidney disease, high blood pressure, and stroke -- conditions whose effects may be irreversible and potentially fatal. Correctly forecasting diabetes using only simple health metrics could lead to a dramatic reduction in long-term detriment to a person's health by inciting them to get early treatment and diagnosis. Because the metrics in this study are very easy to obtain, a method of predicting diabetes using these attributes alone would be especially helpful to historically marginalized groups with little access to advanced medical institutions.

Implementation

All three algorithm implementations were performed using the ABAGAIL java library. The file runTests.java was created by adapting AbaloneTest.java to perform various experiments on the Diabetes dataset. Specifically, an 'iterations' loop was added in order to evaluate the performance of each algorithm as a function of number of iterations. Performance was recorded every 2,500 iterations up to 10,000 iterations.

Two performance metrics were evaluated: 1) % of instances correctly classified, and 2) clock run time. To record the latter, System.nanoTime() was added to record the total time taken for each algorithm to train using *n* iterations.

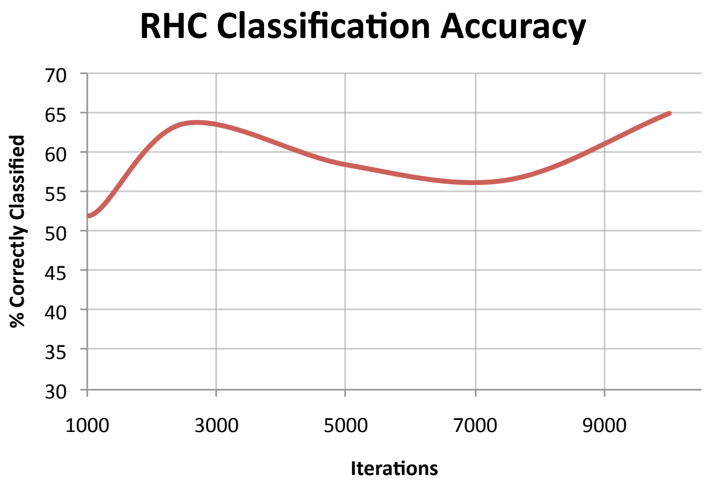
Additionally, the ability to experiment with various hyperparameters was added. For Genetic Algorithm, the ability to test various values for population size, crossover, and mutation rates was added. Similarly, the temperature and cooling rate was varied and tested in the Simulated Annealing experiments.

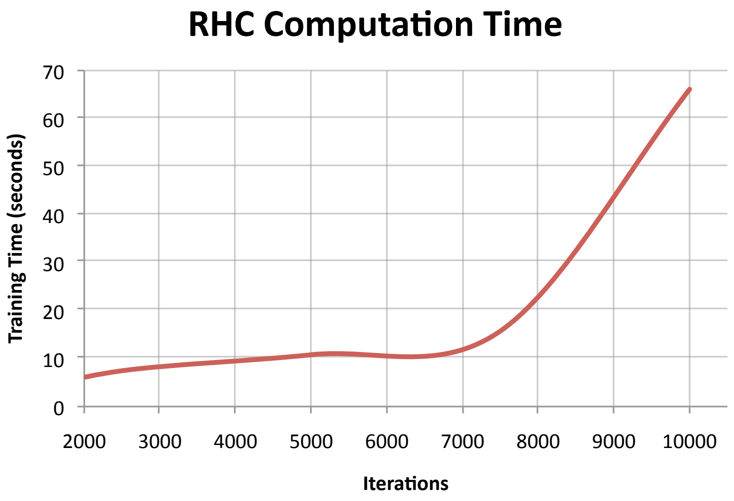
The underlying neural network was configured to match the optimal hyperparameters found in Assignment 1, where the number of input layers was equal to the number of attributes (8), along with 2 hidden layers and 1 output layer.

Randomized Hill Climbing

Randomized Hill Climbing (RHC) is an iterative algorithm that attempts to find a solution by first selecting one at random. Upon each iteration, it seeks to improve the solution by slightly altering an element in the solution. In this case, changing an element corresponds to making a change in one of the weights for the neural network. If the new weights produce a better solution, the solution is adopted and the alteration process continues until no further improvements can be made. In the case of optimization, this corresponds to finding a local optimum. The key drawback of RHC, however, is its inability to guarantee finding a global optimum. Upon finding a local maximum, the algorithm cannot improve its solution locally any further, so it will halt and return a potentially sub-optimal solution. To help alleviate this, RHC performs random re-starts along the solution space in an effort to find various local optima, and hopefully the global optimum as well.

Applied to the Diabetes dataset, we observe the following performance metrics over 10,000 iterations:



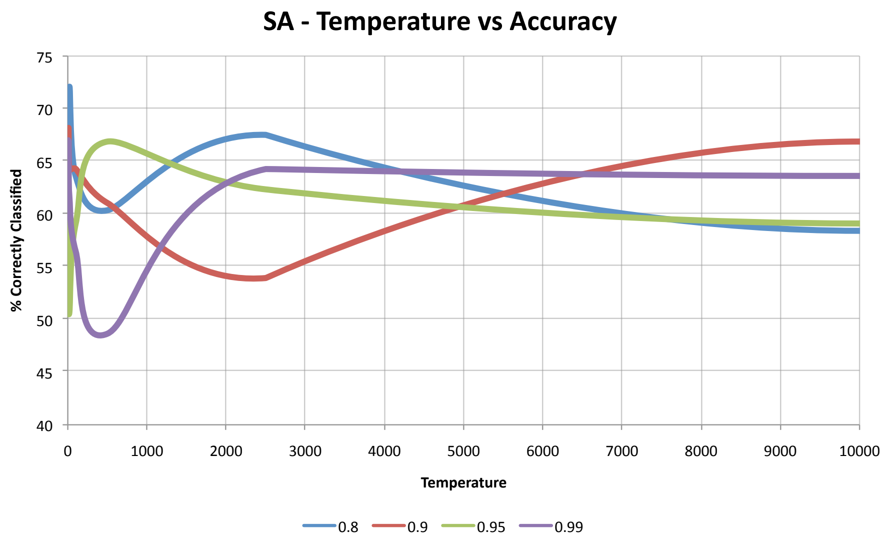


A rapid increase in classification accuracy is observed between the 1000th and 3000th iteration, where an accuracy of 63.6% on the test set is reached. Interestingly, accuracy then decreases for the following 5,500 iterations, possibly indicating that RHC progress was hindered by stagnation around a local optimum. With 10,000 iterations, the accuracy once again reaches an optimum of 64.9% on the test set, signaling a breakout from the prior plateau. Looking at the Computation Time graph, an important difference is noticed between the two local optima found at iteration 2,500 and 10,000. For the former, a training time of 7.29 seconds is found, whereas the latter requires a training time of 64.94 seconds. Given the significant difference in runtime for the small increase in accuracy, training for 2,500 iterations seems to be the most efficient run for RHC.

Simulated Annealing

Simulated Annealing resembles RHC by searching the solution space for many local optima in hopes of finding the global maximum. However, it differs by adopting a probabilistic method for moving to a different state. SA employs a cooling factor that affects the probability for accepting a worse solution. Using this technique, SA improves its likelihood of searching a large solution space while avoiding getting stuck at local optimum. As its temperature decreases, the breadth of its search also decreases and begins to resemble a standard hill climbing algorithm. While finding the global optimum is not guaranteed, simulated annealing provides a methodical approach with which there is good likelihood of finding the best solution.

Applied to the Diabetes dataset, I tested the change classification accuracy as a function of temperature. This experiment was repeated with various cooling factors in order to find optimal hyperparameters for SA using 10,000 iterations:

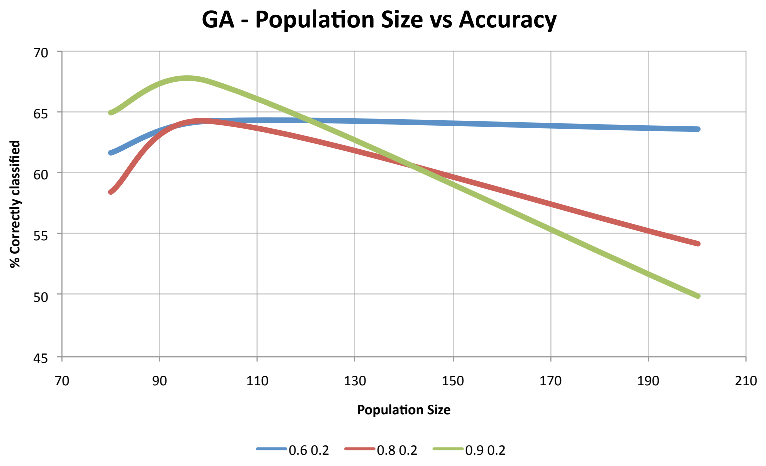


Classification accuracy is highly volatile when temperature is less than 1,000, after which each cooling factor curve varies little and begins to converge. It is interesting to note that the cooling factor of 0.9 performs significantly worse than all 3 other factors when the temperature is at 2,500, but steadily increases to achieve a higher classification accuracy at a temperature of 10,000. Thus a cooling factor of 0.9 with temperature of 10,000 produced the best results by resulting in a classification accuracy of 66.83%. This likely implies that having too high of a cooling factor will discourage successive iterations from venturing into distant local optima.

Genetic Algorithm

Genetic Algorithms (GA) generates a pool of hypotheses, typically bit strings, and aim to find the one that optimizes fitness by iteratively updating the population of hypotheses. On each iteration, GA evaluate the fitness of all members of the population, upon which it selects the most fit individuals to generate a new population. Some of these are carried directly, while others are created using crossover and mutation to generate diversity in offspring. In doing so, each iteration contains a population whose fitness gets progressively closer to the optimal solution, in theory. An important drawback of GA is that in using the most fit individuals to generate offspring, ensuing populations may become increasingly homogeneous. This problem of crowding is analogous to RHC's problem of getting stuck at local optima.

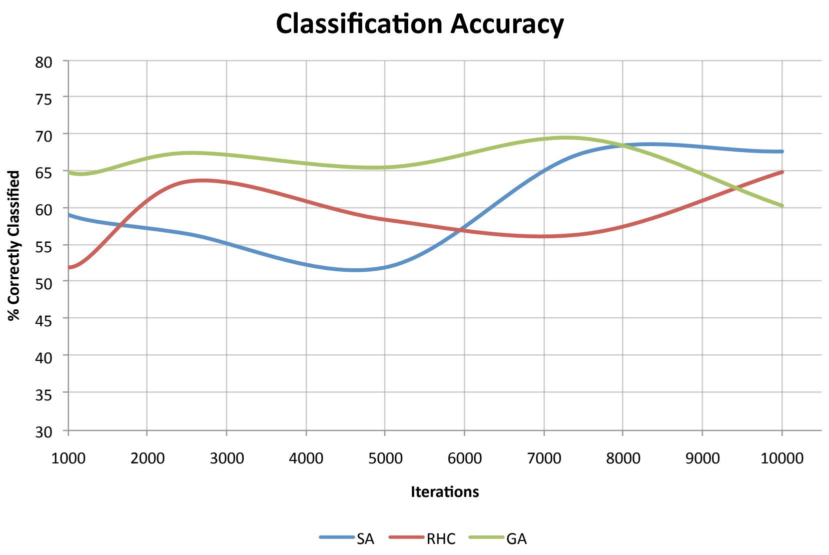
To examine the effect of population size on classification accuracy, I ran the below experiment using 10,000 iterations and various values for crossover and mutation.

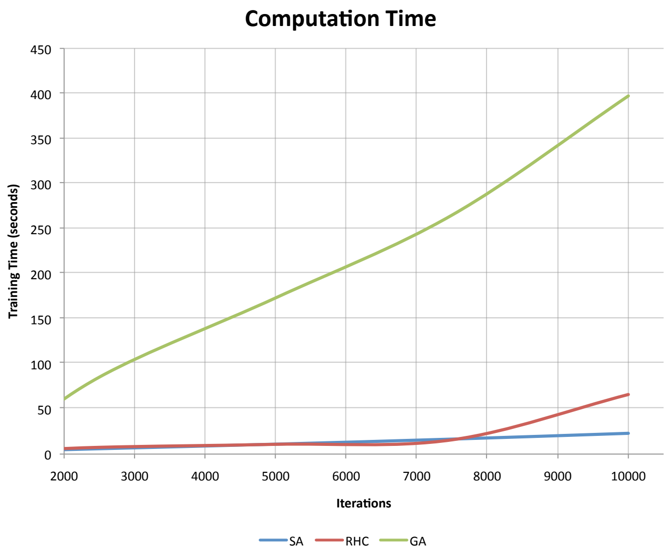


As seen above, classification accuracy increases rapidly as the population is increased from 80 to 100, after which it steadily decreases. This leads me to believe that having too small of a population size results in little diversity and a lesser chance of it containing the optimal hypothesis. On the other hand, too large of a population size will lead to too many hypotheses being considered and will leave little hope in making progress towards finding the optimal fitness.

Algorithm Comparison

Using the optimal parameters found for each of the three algorithms, I ran RHC, SA, and GA on the diabetes test set in order to compare their relative performance in terms of classification accuracy as well as computation time.





From the Classification Accuracy graph, we can see that GA is the best performer through 7,500 iterations, reaching a maximum accuracy of 69.48% when run with 7,500 iterations. Looking at the Computation Time graph, however, it can be seen that this comes with a great cost: 265.27 seconds of training time for a training set size of only 469 instances. Relative to the other two algorithms, GA's runtime seems one of its greatest liabilities. Simulated Annealing, which correctly classified 67.53% of instances using 10,000 iterations did so with a training time of only 23.05 seconds, over 10 times faster than GA.

From Assignment 1, the accuracy of the Neural Network using backpropagation with 8 input layers, 2 hidden layers, 1 output layer along with a learning rate of 0.3 and momentum of 0.2 resulted in a classification accuracy of 76.55%. This leaves backpropagation as the best performer compared to all three randomized algorithms.

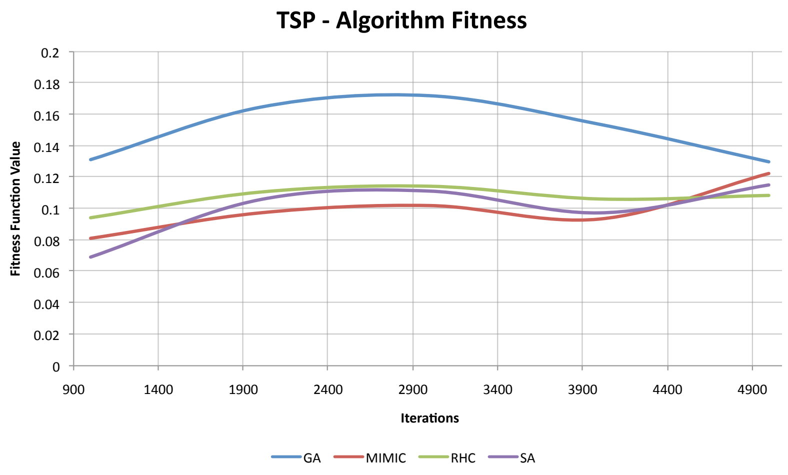
Part 2: Optimization Problems

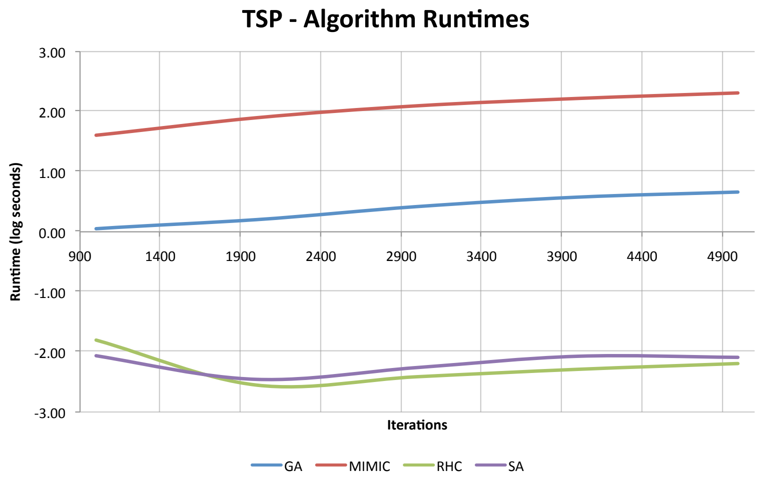
Three optimization problems were selected to highlight the differences and strengths of Genetic Algorithms, Simulated Annealing and MIMIC.

Traveling Salesman Problem

The TSP presents N cities with and distances between each pair of cities. The traveling salesman wishes to complete the circuit by visiting each city exactly once, and do so with minimal cost- in this case distance. Although the problem description is quite simple, finding an optimal solution is proven to be NP hard. Since no known algorithm is known to solve TSP fast, it would be interesting to apply randomized approaches and gauge their respective success in solving this classic problem in computer science.

The evaluation of the four algorithms was conducted by comparing number of iterations to the fitness function value. I performed up to 5,000 iterations and computed the runtime for each algorithm.





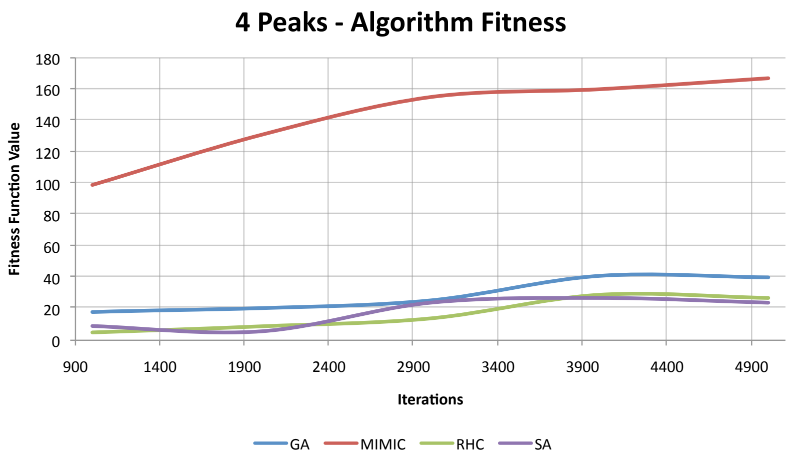
The experiment readily reveals the superiority of the Genetic Algorithm in finding an optimal solution, reaching a fitness value of 0.17 using 3,000 iterations, a 154% improvement from the next best performer (RHC). Given the potentially enormous number of path permutations in the search space, RHC and SA are likely to be plagued by and obsess over local optima, making little progress from one iteration to the next. MIMIC also encounters difficulty in this type of search space. By evaluating solutions based on the probability that a global maximum exists in a certain area, MIMIC is likely to believe that paths similar to each other will have similar fitness. This is not a safe assumption to make in the TSP, where a one path difference could lead to a large change in distance.

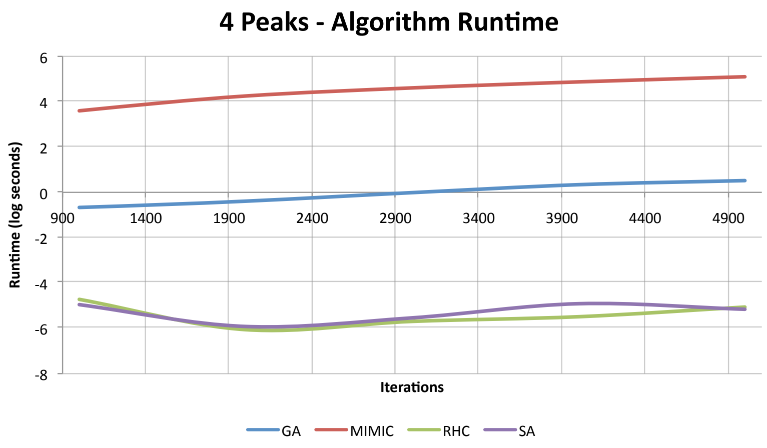
GA's performance is due in large part to the way it searches through the hypothesis spaces. By taking the best performing hypotheses and altering some of them slightly across iterations, GA methodically generates a more fit population of paths each iteration. Crossover and mutation allow GA to make slight alterations to already good solutions, allowing it to build upon prior iterations.

Four Peaks Problem

The four peaks problem (FPP) presents an interesting puzzle for randomized search algorithms. FPP uses a function with input N and a trigger position T to generate a bit string which contains two global maxima, located either when T+1 leading 1's are followed by all 0's or when T+1 trailing 0's are preceded by all 1's. The two other local optima are located with a string of all 1's or all 0's. As the value of N grows large, it becomes more difficult to find the basin containing the true global maxima.

This experiment was run by testing the four algorithms with up to 5,000 iterations and comparing their performance in runtime and fitness value achieved.





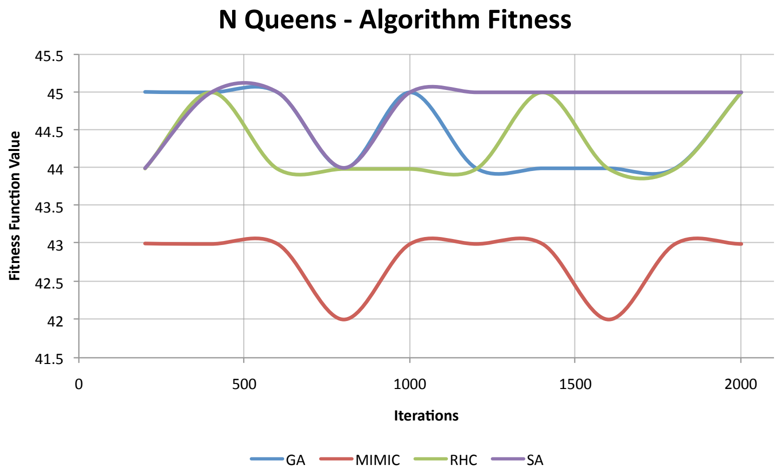
Given the large basin of attraction around the local optima, I predicted RHC and SA to encounter problems exiting these points and finding the global optima. As summarized above, this proved to be the case indeed. Random restarts performed by randomized hill climbing likely had a high probability of landing in a 'false' basin and staying in it. Similarly, SA seems to have encountered a high rate of getting trapped in the false basins. Given that this is the prime weakness of RHC and SA, it is not unexpected to have found them the two worst performers in the FPP.

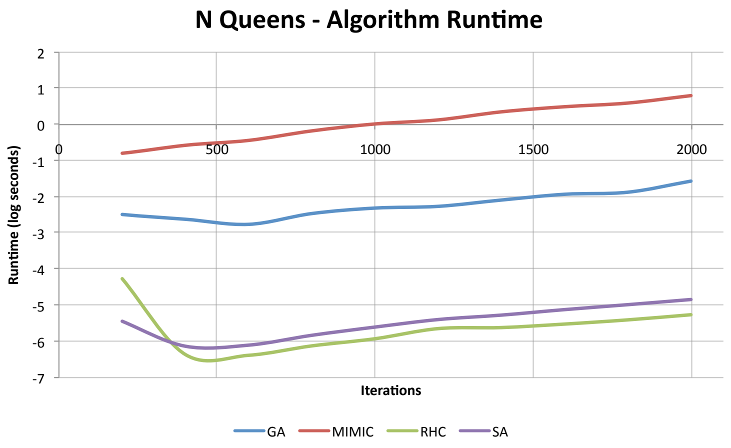
Most notably, the fitness function value achieved by MIMIC far outweighs the other three competitors: 167 using 5,000 iterations, a 400% improvement over the next best algorithm (GA). On every iteration, MIMIC reduces its search space by focusing exclusively on the subspace wit the highest probability of containing a global maximum. With four maxima to consider, MIMIC will quickly eliminate non-optimal maxima and narrow its search to the peaks containing the two sought-after peaks. By using this probability density-based structure, MIMIC builds upon prior findings to make progress, as opposed to selecting random restarts after each iteration. The cost to this effectiveness is its computation time: MIMIC takes significantly longer than the other algorithms to achieve its results. In larger scale optimization problems with limited time resources, runtime may be one of MIMIC's most significant drawbacks.

N Queens Problem

The N Queens Problem (NQP) involves placing N queens on a NxN chessboard such that no queen is attacked by any other queen. In the context of optimization, the aim is to minimize the number of queens that attack each other. Since there may be more than one configuration where no queen is in danger, there will be multiple global optima in the solution space. Similarly, local optima represent configurations where only 2 queens are attacked, for example.

The four algorithms were applied to this problem using up to 2,000 iterations. Once more, the performance was measured in terms of fitness function value and overall runtime.





Simulated Annealing proved to be the most consistent performer with highest fitness value, reaching a value of 45 on 500, 1000, 1500, and 2000 iterations. Making its case for the algorithm of choice for NQP, it also had the second lowest runtime- about 3 orders of magnitude faster than GA and 5 orders faster than MIMIC.

Randomized Hill Climbing also reached this maximum value of 45, though it did so only on iteration counts of 400, 1400, and 2000. In this problem, the position of the N queens on the board corresponds to a hypothesis. Given this search space, a hill climbing algorithm would alter the position of a queen only if doing so results in fewer collisions. Consequently, hill climbing based algorithms could fall victim to reaching a solution whose collisions are not optimal but for which an alteration makes no improvement. Randomized restarts can thus come in and alleviate this issue, allowing the algorithm to reach a global optima with enough random restarts.

Given the strong resemblance to RHC, Simulated Annealing can be expected to perform similarly. Given how it outperformed RHC, however, it appears as though the temperature and cooling features proved advantageous. By cooling the temperature gradually, SA can encounter the various local optima early in the process, and move towards states of lower energy until a global optima is found.