N-queens puzzle

The **n-queens puzzle** is the problem of placing n chess queens on an n×n chessboard so that no two queens threaten each other; thus, a solution requires that no two queens share the same row, column, or diagonal. The solution to n-queen problem exists for all natural numbers *n* with the exception of *n* = 2 and *n* = 3. The problem of finding all solutions to the n-queens problem can be quite computationally expensive, as there are nxnCn possible arrangements of n queens on a n×n board. However, by applying a simple rule that constrains each queen to a single column or row and generating permutations, it is possible to reduce the number of possibilities to n!, which are then checked for diagonal attacks. Thus n-queens problems is a really good candidate to compare different randomised search algorithms. A brute force algorithm would have taken 40320 iterations for finding optimal value in case of 8 queens problems. We compare the performance of 4 different randomised algorithms i.e. Randomized Hill Climbing, Simulated Annealing, Genetic Algorithm and MIMIC on 8-queens problem. We also tuned different hyper-parameters to compare fitness value and time taken to execute the algorithms. We kept maximum iterations to 1000

**Randomised Hill Climbing:** For n-queens problem, we kept the value of n to 8. We also tuned max attempts parameter of randomised hill climbing algorithm while keeping max iterations to 1000 and restarts = 0. It is obvious that time taken to complete the run was increasing for increasing value of max attempts. The max attempts values of 10, 15, 20, 25, 30, 35, 40 were not able to find optimal value. For 8 queens problems, the randomized hill climbing algorithm was able to find optimal value with max attempts of 45 and above in only 80 steps to find best state of [4 2 7 3 6 0 5 1]. This is extremely faster than brute force approach.

|  |  |
| --- | --- |
| Chart, line chart  Description automatically generated | Chart  Description automatically generated |

In order to get out of local minima, the randomized hill climbing algorithm uses random restarts which essentially mean starting algorithm with some random point. Random restart was not required in this case.

**Simulated Annealing:** For n-queens problem, we tried to tune value of max attempt (i.e. maximum number of attempts to find a better neighbours at each step) for both exponential schedule and geometric schedule for the temperature parameter. For exponential decay for temperature parameter, the algorithm didn’t converge for max attempts values of 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100 whereas with the geometric decay, it converged for the value of 70 and above. The best state found was [4 7 3 0 2 5 1 6].

|  |  |
| --- | --- |
| SA with Exponential Decay tuning max attempts | SA with Geometric Decay tuning max attempts |
| Chart, line chart  Description automatically generated | Chart, line chart  Description automatically generated |

**Genetic Algorithm:** For 8-queen problem, we tuned max attempts, mutation probability and population size parameters separately while keeping other parameters constant. The best set of value for max attempt, population size and mutation probability was 10, 50 and 0.1 respectively. The best state obtained was [3 5 7 2 0 6 4 1].

|  |  |  |
| --- | --- | --- |
| GA Max Attempts vs Fitness | GA Mutation Probability vs Fitness | GA Population size vs Fitness |
| Chart, line chart  Description automatically generated | Chart, line chart  Description automatically generated | Chart, line chart  Description automatically generated |

**MIMIC:** For 8-queen problem, we tuned max attempts, keep percentage and population size separately. The best set of parameters have values of max attempts, population size and keep pct of 5, 80 and 0.10. It is visible from the graphs that max attempts didn’t have any impact on fitness value. For population size of 80, fitness function had lowest value of 80. The keep percentage value of 0.10, 0.20 and 0.30 produced fitness value of 1. The 8-queen problem didn’t converge by using MIMIC algorithm. The best state found was [3 0 6 1 2 5 7 4].

|  |  |  |
| --- | --- | --- |
| MIMIC Max Attempts vs Fitness | MIMIC Population Size vs Fitness | MIMIC Keep Pct vs Fitness |
| Chart, line chart  Description automatically generated | Chart, line chart  Description automatically generated | Chart, line chart  Description automatically generated |

**Comparison of different algorithms:**

|  |  |  |
| --- | --- | --- |
| Fitness wrt iterations of different algorithms | Execution time of different algorithms | Algorithms and Fitness Score |
| Chart, histogram  Description automatically generated | **Chart, bar chart  Description automatically generated** | **Chart, bar chart  Description automatically generated** |

Travelling salesman problem

The **travelling salesman problem**(also called the **traveling salesperson problem**[[1]](https://en.wikipedia.org/wiki/Travelling_salesman_problem#cite_note-1) or **TSP**) asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?" It is an NP-hard problem.

**Randomised Hill Climbing**: We applied randomised hill climbing tuning max attempts and restart parameter separately by trying different values keeping max iterations to 1000. The minimum fitness value of 17.38 is achieved for max attempts of 40 and above. The fitness value didn’t change with restart parameter. Below graphs shows relationship between max attempts and fitness value and restart and fitness value.

|  |  |
| --- | --- |
| Randomized Hill Climbing with varying max attempts | Randomized Hill Climbing with varying restarts |
|  | Chart, box and whisker chart  Description automatically generated |

**Simulated Annealing:** We tuned max attempts parameter for simulated annealing for exponential and geometric decay. The exponential constant was set to 0.005 while decay factor in geometric decay was set to 0.99. For both, the minimum fitness value was found to be 17.34, however fitness value was highly dependent on max attempts while it was constant for geometric decay of temperature.

|  |  |
| --- | --- |
| SA with varying max attempts exp. decay | SA with varying max attempts geometric decay |
|  | **Chart  Description automatically generated** |

**Genetic Algorithm:** For genetic algorithm, we tuned max attempts while keeping population size, mutation probability and max iterations to 100, 0.1 and 1000 respectively. For max attempts, the fitness value is found to be constant where minimum fitness value was 17.34. Similarly, we tuned mutual probability while keeping population size, max attempts and max iterations values to 100, 10, 1000. The fitness value of mutation probability is found to be constant. It also achieved a value of 17.34. Likewise, the population value is also tuned which keeping max attempts, mutation probability and max iterations value to 10, 0.1 and 1000. The fitness value decreased as population value increased from 10 to 40 to reach 17.34 and thereafter it remained constant with the same value.

|  |  |  |
| --- | --- | --- |
| GA with varying Max Attempts | GA with varying mutation probability | GAs with varying population size |
|  |  |  |

**MIMIC:** We tuned max attempts, population size and proportion of samples to keep at each iteration separately while keeping other parameters value constant. The fitness value didn’t reach optimal value for different values of max attempt and it clear for below graph that fitness value is not affected by max attempts value. Also, fitness value decreased for increasing value of population size. The MIMIC value achieved minimum fitness value of 17.34 for population size, max attempts and keep pct of 80, 50, 0.40 respectively**.**

|  |  |  |
| --- | --- | --- |
| MIMIC with varying Max Attempts | MIMIC with varying population size | MIMIC with varying keep percentage |
| **Chart  Description automatically generated** |  | **Chart, line chart  Description automatically generated** |

**Comparison on different algorithms:**

|  |  |  |
| --- | --- | --- |
| Fitness wrt iterations of different algorithms | Execution time of different algorithms | Algorithms and fitness Score |
| **Chart, line chart  Description automatically generated** | **Chart, bar chart  Description automatically generated** | **Chart, bar chart  Description automatically generated** |

**4 Peaks problem**

Evaluates the fitness of an n-dimensional state vector *x*, given parameter *T*, as:  
*,*

**Randomised Hill Climbing:** The randomised algorithm got trapped into local minima, even after attempts were made to take it out of local minima using random restarts it was still getting stuck at local minima. The maximum value of fitness function it achieved was 40 even after tying different values of max attempts and random restart.

The below figure provides an idea even after trying different values of max attempts and restarts, the fitness is not changing.

|  |  |
| --- | --- |
| Randomised Hill Climbing with varying max attempts | Randomised Hill Climbing with varying restarts |
| Chart  Description automatically generated | Chart  Description automatically generated |

**Simulated Annealing:** The performance of simulated annealing was more or less same for different values of max attempts both in case of exponential decay for temperature and geometric decay for temperature which suggest that it got trapped in the local minima. The problem persisted even after trying different values on exponential constant in exponential schedular and decay factor in geometric schedular.

|  |  |
| --- | --- |
| SA with varying max attempts with exponential decay | SA with varying max attempts with geometric decay |
| Chart, line chart  Description automatically generated | Chart, line chart  Description automatically generated |

**Genetic Algorithm:** We tuned different values max attempts, mutation probability and population size separately with keeping other parameters constant. For max attempt of 10 and above keeping population size 100, mutation probability of 0.1 and max iterations of 1000, genetic algorithm was able to find near optimal value and was able to get out of local minima. Below figure provides fitness scores across different values of max attempts, mutation probability and population size. The best set of values for max attempts, mutation probability and population size are 10, 0.1 and 100 respectively. The best fitness score achieved was 75.

|  |  |  |
| --- | --- | --- |
| GA with varying Max Attempts | GA with varying mutation probability | GA with varying population size |
| **Chart  Description automatically generated** | **Chart, line chart  Description automatically generated** | **Chart, line chart  Description automatically generated** |

**MIMIC**: Even though MIMIC was not close to near optimal value, it performed better than Randomised Hill climbing and simulated annealing. For different values of max attempts, the fitness score was constant. The best fitness score MIMIC was able to achieve was 55 for max attempts of 10, population size of 80 and keep percentage of 0.25.

|  |  |  |
| --- | --- | --- |
| MIMIC with varying Keep Percentage | MIMIC with varying Max Attempts | MIMIC with varying population size |
| **Chart, line chart  Description automatically generated** | **Chart  Description automatically generated** | **Chart, line chart  Description automatically generated** |

**Comparison on different algorithms:**

|  |  |  |
| --- | --- | --- |
| Fitness wrt iterations of different algorithms | Execution time of different algorithms | Algorithms and fitness Score |
| **Chart, line chart  Description automatically generated** | **Chart, bar chart  Description automatically generated** | **Chart, bar chart  Description automatically generated** |

**Comparison of randomised optimization algorithms and gradient descent on neural network weight optimization problem:**

**Dataset:** **BNP Paribas Cardif Claims Management:**

This dataset has 131 predictors where 113 predictors are numerical and 19 are categorical with varying cardinality. Also, 33.5% of the dataset has missing values. Owing to high no. of predictors columns with both numerical and categorical columns, large set of missing values, highly correlated and right skewed features made this dataset challenging and exiting to compare performance of different optimization algorithms for neural nets optimization. The target column is the binary column and the objective to predict if a sample belongs to category 0 or 1. The target column is also imbalance with 76% samples belong to category 1. We used f1 score to compare performance.

**Pre-processing:** The dataset has around 33.5% of missing values. We used -1 to replace all missing values in numerical columns and “\_\_MISS\_\_” to replace missing values in categorical columns. The dataset has 19 categorical, hence conversion of categorical columns to numerical is required as mandated by most of the ML algorithms. One standard approach of representing the categorical variables is One hot encoding, however, owing to high cardinality in some of the categories which in effect would induce high dimensionality in the data, we used Target encoding with Laplace prior with smoothness value of 300. Since the dataset contains large number of predictors it is advisable to apply feature selection algorithms to remove unwanted and noisy features. We applied Boruta feature selection algorithm to remove irrelevant features (Note that: comparison of features selection algorithms is not in the scope of this study). After applying Boruta, we nailed down to 23 most relevant features and discard rest of others. We also scaled the predictors to effectively apply optimization algorithms. We had divided the dataset into training and validation in the ratio of 70:30 with stratified sampling in-order to maintain distribution of data between class 1 and class 0 in the validation sets.

Randomised Hill Climbing: We applied randomised hill climbing and tuned max attempts parameter of hill climbing along with learning rate. We tried max attempts = [50, 100] and learning rates = [0.001, 0.005, 0.01, 0.1].