8-Queens Problem

Pratham Aggarwal

Abstract--- This report presents an evolutionary algorithm approach to solving the 8-Queens problem, a classic optimization problem, and evaluates the algorithms performance in terms of avoiding local optima and determining the global maximum by adjusting important parameters.

1 INTRODUCTION

This report focuses on solving the 8-queens problem, a well-known optimization problem, using the metaheuristic algorithm. It shows the algorithm's distinct behavior when the parameters are changed, as well as how it handles one of the primary challenges by these types of problems.

1.1 Motivation

Metaheuristic algorithms use natural principles to solve optimization problems. This method evolves through time, attempting to improve each time until it finds an acceptable solution. Studying how animals adapt and evolve contributes to the creation of systems that continuously learn and improve, thereby opening up possibilities for solving real-world problems. The 8-queens problem, as an optimization problem, is an excellent example for testing this type of techniques.

1.2 Problem Statement

Solving 8-queens problem using evolutionary algorithms. Analyzing the performance of algorithm to avoid local optima and find the best fitness score possible to get an arrangement of placing 8 queens on boards without any conflicts.

1.3 Related Work

Backtracking is one of the most well-known methods for solving the 8-queens problem. This method uses an algorithm to evaluate all possible solutions and determine whether or not each solution meets the problem requirements. Another option is to solve the problem using a metaheuristic algorithm, such as Genetic Algorithm or Simulated Annealing. Using metaheuristic algorithms allows us to avoid getting trapped in local optima and find a balance between exploration and exploitation.

1.4 Contributions

In our project, we tried to adjust the parameters of the algorithm and analyze its performance using a convergence graph. We decided on the default parameters that were provided to us and increased and decreased them accordingly. We analyzed the relationship between fitness value and iteration count to demonstrate the algorithm's performance. Changing parameters produced varying results, showing which parameters help improve and impact algorithm performance.

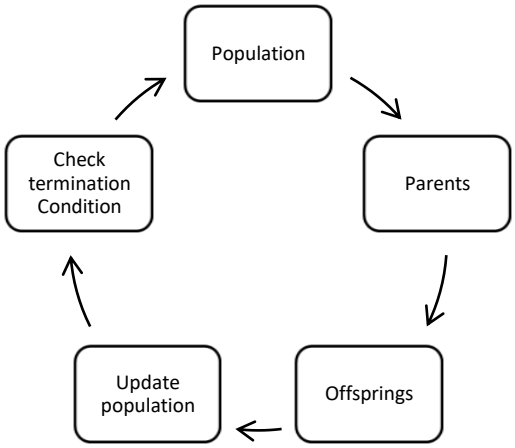

Figure 1: Approach flowchart

2 Background Material

Different terms we used:

Population Size: It represents the potential solution(attempts) in a solution space. Each attempt represents the possible positions of the queens on the board

Initialization: The initial population is randomly generated

Parent Selection: Selecting randomly 5 parents (attempts) from the population

Recombination: (Cut-and-Cross-fill) The crossover operation to generate the offsprings from parents.

Mutation: (Swap) The process of mutating the offsprings generated. It provides diversity in the solution space.

Mutation Probability: The probability with which the mutation occurs.

Fitness score: The fitness score represents how good or optimal a solution attempt is in the population. A higher fitness score indicates a better solution.

Termination Condition: (Iteration Limit) The condition where the algorithm terminated and gives the best score.

3 Approach

Our approach is primarily based on the flowchart shown in Fig 1. We begin by randomizing the population. In each iteration, we choose random attempts as parents. Then we do a crossover on the first two parents to produce offsprings. The crossover takes place by identifying a random crossover point and swapping the chromosomes of the two parents around that point, resulting in new offspring. After the offsprings were generated, we carried out swap mutations on them with a particular probability. For this, we chose two random indexes and swapped their values, thus swapping the positions of two queens chosen at random. This helped us incorporate diversity in the population. After we have offsprings, our next step is to replace them with the worst solutions in the population. To identify the worst attempts in the population, we calculated the fitness score for each attempt and to calculate the fitness score for each attempt, we first determined the penalty for each attempt. The penalty is calculated using the number of conflicts for each queen in the attempt. The penalty for an individual queen in a particular position (row and column) is determined by adding the conflicts with other queens in the same row, column, and diagonal. The penalty calculator iterates over all the other queens (except the current queen) and increases the penalty count if there is a conflict. The diagonal is checked by the absolute difference between the row numbers is equal to the absolute difference between the column numbers. The penalty function calculates the total penalty for each queen in the attempt and returns it. The fitness score for the attempt is then computed as the inverse of penalty + 1. Adding one allows us to avoid dividing by zero for the attempt with zero penalty. So, the attempts with the highest fitness score, maximum-1 (showing no conflicts found), have the lowest penalties. Finally, we replace the attempts having the lowest fitness scores. This process will continue until we reach a termination limit. We ran the algorithm for 100 trials, and the average fitness score for each iteration was calculated across all trials. The best attempt found in the last trial is displayed along with the convergence plot.

Experiment set-up

Different values we used to test our algorithm:
Fitness score = 1 / (penalty +1)
Number of parents: 5
Population Size: 100 (Default), 500, 50
Termination Limit: 10000(default), 20000, 100
Offspring count: 2 (Default), 1, 50
Mutation Probability: 80% (Default), 99.9%, 20%
Trials: 100

3.1 Results and Discussion

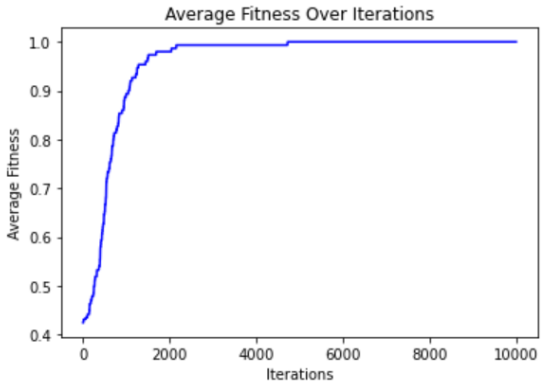

Figure 2 : Average Fitness Vs Iterations

The convergence graphs showed different behavior depending on the parameters. With the default parameters (Fig 2), the fitness value increased steadily and consistently in the early iterations, indicating exploration of the space followed by solution exploitation. The absence of major plateaus and early convergence indicated that the algorithm avoided being stuck in local optima. Then, increasing the mutation probability allowed for greater space exploration before finding the global maximum, whereas decreasing it resulted in early trapping in local optima, resulting in the absence of global maximum. Next, increased offspring count resulted in significant computational overhead, while decreasing it had low effect. Increasing population size considerably increased mean fitness value exploration and exploitation, but decreasing it minimized later exploitation but still found the answer. Lastly, on increasing the termination limit, we saw that it attempted to balance exploration and exploitation in the first few iterations but found the solution quickly, whereas on decreasing, we saw many plateaus, indicating that the algorithm was trapped in local optima and was unable to find global maximum.

4 Conclusions

The experiments and results show that the evolutionary algorithm is effective at solving the 8-Queens problem. We learnt a lot about the algorithm's behavior and its ability to balance exploration and exploitation by

evaluating the convergence graphs and the impact of different parameters. It highlights the significance of adjusting parameters in metaheuristic algorithms in improving their performance and increasing the possibility of balancing exploration and exploitation. In future we aim to consider more different values of each parameter and get different convergence graphs. Overall, the evolutionary algorithm is a strong and adaptable method to solving complicated optimization problems showing its potential for use in a variety of real-world problems.

5 References

https://research.ijcaonline.org/volume102/number7/pxc3898667.pdf

https://educative.io/answers/solving-the-8-queen-problem-using-genetic-algorithm

https://medium.com/nerd-for-tech/genetic-algorithm-8-queens-problem-b01730e673fd

https://www.researchgate.net/publication/294121515_Solving_8-Queens_Problem_by_Using_Genetic_Algorithms_Simulated_Annealing_and_Randomization_Method